UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA DE TÉCNICA DE TELECOMUNICACIÓN:

SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

ALGORITMOS ADAPTATIVOS DE NÚCLEOS DE MERCER

Autor: MIGUEL ÁNGEL SANZ MUÑOZ
Director: DR. MANEL MÁRTINEZ RAMÓN
Tutora: DRa. VANESSA GÓMEZ VERDEJO

MARZO DE 2014

No podemos resolver problemas
pensando de la misma manera
que cuando los creamos.
Albert Einstein.

Agradecimientos

Quiero expresar mi agradecimiento en primer lugar a mis padres Miguel y Benita, a mi hermano Javier y a mi novia Hannah, por el apoyo, los consejos y la ayuda que me han dado a lo largo de la carrera y de este proyecto fin de carrera.

También quiero dar las gracias en especial a mi tía Conchi por ayudarme a comprender y entender las matemáticas en estudios no superiores, que ha hecho que me gusten tanto y que con su ayuda tuviera la base que ahora tengo y que me han sido de gran utilidad en mis estudios superiores.

Agradezco a mis compañeros y amigos de la universidad con los que he pasado muy buenos momentos y por la ayuda que nos hemos prestado mutuamente, por lo que quiero dar las gracias a Mach, Edu, Luis, Natalia, Marta y Jorge.

Gracias a mis amigos de siempre por su apoyo y por los buenos momentos pasados con ellos que me ayudaron a desconectar y disfrutar de la vida.

Finalmente, quería agradecerles a mis profesores de la universidad por todo lo que me han enseñado sobre antenas, radiotransmisión, estadística, electrónica, física, matemáticas y programación. Y a mi tutor Manel, por sus conocimientos en el tratamiento de datos y los algoritmos de aprendizaje.

Muchas gracias a todos por vuestra ayuda.

ÍNDICE GENERAL

CAPÍTULO 1

INTRODUCCIÓN

El objetivo de este Proyecto Fin de Carrera es la introducción del kernel recursivo en los algoritmos lineales de kernel, en nuestro caso utilizaremos los algoritmos KLMS y KRLS, para su aplicación a los sistemas de comunicación como es en nuestro caso.

Dichos algoritmos serán usados en el procesado de un arrays de antenas y observaremos cómo se comportan en diferentes escenarios.

El array de antenas utilizado para estas pruebas, consta de un array lineal de 7 elementos, dichos elementos son antenas de tipo dipolo que trabajan a una frecuencia de 3 GHz, usando una modulación BPSK y están separados entre si una distancia de $\lambda/2$.

Además, tendremos que tener en cuenta que al tratarse de antenas dipolo se producirán varios efectos como es el acoplamiento mutuo de las señales entre los diferentes elementos. También tendremos que tener en cuenta la conformación del haz de la antena, hacia donde esta direccionada y las posibles diferencias temporales de las señales recibidas debido al multitrayecto.

Utilizaremos un conformador de haz que se encargara de minimizar la probabilidad de error al detectar la señal de entrada. En nuestro caso utilizaremos un conformador de haz con referencia temporal adaptativo. Esto es debido a que al minimizar la probabilidad de error utilizaremos los algoritmos lineales de kernel ya mencionados, sobre una matriz de kernel recursiva.

Para finalizar compararemos el comportamiento de estos algoritmos sobre la matriz de kernel recursiva y sobre la matriz de kernel no recursiva. De esta comparación esperamos comprobar que al usar la matriz de kernel recursivo, minimizamos la probabilidad de error que obteníamos sin usar el método recursivo.

CAPÍTULO 2

PROCESADO LINEAL DE ARRAYS

2.1. Conformación de haz

El procesado lineal de arrays es el conjunto de técnicas utilizadas para conformar un haz de apuntamiento a una antena de forma que se reciba una señal procedente de una dirección de llegada, minimizando las interferencias procedentes de otros ángulos de llegada mediante algún tipo de optimización.

El conformador de haz es lineal si el proceso de detección se puede expresar mediante la función $y[n] = \mathbf{w} \cdot \mathbf{x}[n]$, donde $y[n]$ es la señal de salida y $\mathbf{x}[\mathbf{n}]$ es la señal de entrada. El algoritmo necesario para encontrar los pesos $\mathbf{w}$ es lineal si este conjunto de pesos se puede expresar como una combinación lineal de la señal de entrada procedente de la antena.

El conformador de haz es no lineal cuando la salida $y[n]$ no se puede expresar cómo una combinación lineal de la señal de entrada, $\mathbf{x}[n]$ y los pesos obtenidos.

Nosotros queremos construir un conformador de haz no lineal utilizando un método para transformar la señal de entrada $\mathbf{x}[n]$ a un conjunto de datos lineales transformados a una dimensión superior.

Este conformador de haz no es lineal pero al usar el conjunto de datos transformados en el espacio de Hilbert, podemos utilizar sus propiedades propias. Dichas propiedades nos permiten usar el conformador como si fuera un sistema lineal. Por

eso podemos aplicar a nuestro conjunto de datos transformados los algoritmos de aprendizaje lineales Least Mean Squares (LMS) y Recursive Least Squares (RLS).

Se denominan dos tipos de conformador de haz: el conformador de haz con referencia espacial y el conformador de haz con referencia temporal.

2.2. Conformación de haz con referencia espacial

Es el tipo de conformador de haz o procesamiento de la señal que utilizaremos cuando conocemos el ángulo de entrada de la señal y también conocemos la señal de entrada al conformador de haz, $\mathbf{x}[n]$. Pero en este caso desconocemos la señal de salida deseada del conformador, $d[n]$.

Sabemos que los datos estimados de la salida del conformador de haz se calculan de la siguiente manera.

$$y[n] = \mathbf{w}^H \mathbf{x}[n] + e[n] \tag{2.1}$$

Para calcular la probabilidad de error lo que tenemos que minimizar es

$$minE\{\|\hat{d}[n]\|^2\} = E\{\|\mathbf{w}^H \mathbf{x}[n]\|^2\} = \mathbf{w}^H \mathbf{R} \mathbf{w} \tag{2.2}$$

donde $\mathbf{R}$ es la matriz de correlación espacial de la señal de entrada, $\mathbf{x}[n]$ y $\mathbf{w}$ son los pesos del algoritmo de aprendizaje utilizado.

Como no conocemos la estadística del conjunto de datos de la entrada para calcular la esperanza, tenemos que calcular una estimación del conjunto de datos que tenemos, $\hat{\mathbf{R}}$.

$$\hat{\mathbf{R}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}^H[n] \mathbf{x}[n] \tag{2.3}$$

Sabiendo que

$$\mathbf{w}^H \mathbf{a_d}(\Omega) = r \tag{2.4}$$

donde $\mathbf{a_d}(\Omega)$ es la entrada deseada de la fuente de la señal para un ángulo de llegada dado, Ω. Para obtener los pesos, despejamos w de la función anterior y obtenemos que

$$\mathbf{w} = \frac{r \mathbf{R}^{-1} \mathbf{a_d}}{\mathbf{a_d}^H \mathbf{R}^{-1} \mathbf{a_d}} \tag{2.5}$$

2.3. Conformación de haz con referencia temporal

Es el tipo de conformador de haz o procesamiento de la señal que será utilizado cuando tenemos como datos conocidos, la señal de entrada del conformador de haz, $\mathbf{x}[n]$ y un conjunto de datos deseados a la salida del conformador. Pero desconocemos el ángulo de entrada de las fuentes de las señales que transmiten.

La salida estimada del conformador de haz se calculará mediante

$$\hat{y}[n] = \mathbf{w}^H \mathbf{x}[n] + e[n] \tag{2.6}$$

Nuestro objetivo óptimo es minimizar la probabilidad de error pero como no podemos calcularlo, lo que lo que hacemos es calcular el mínimo error cuadrático medio, MMSE cuando el ruido es gaussiano blanco ya que es equivalente a calcular la probabilidad de error.

Alternativamente minimizamos la función de coste sobre el error, la cual normalmente es cuadrática, y nos da el siguiente resultado

$$E\{\|e\|^2\} = E\{d[n] - \mathbf{w}^H \mathbf{x}[n]\} \tag{2.7}$$

la función 2.7 la derivamos con respecto a $\mathbf{w}$, y nos dará como resultado lo siguiente

$$\mathbf{w} = \mathbf{R}^{-1} p \tag{2.8}$$

donde p es la correlación cruzada entre el conjunto de datos deseados y la señal de entrada al conformador de haz y se calcula de la siguiente forma.

$$p = E\{\mathbf{x}^H[n]\hat{d}[n]\} \tag{2.9}$$

pero tenemos que calcular la estimación de p, $\hat{p}$, y definir de forma temporal ya que no podemos calcular la esperanza de la señal de entrada debido a que no conocemos la estadística de la señal de entrada. Por lo tanto, tenemos que la estimación de p es

$$\hat{p} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}^H[n] d[n] \tag{2.10}$$

2.4. Conformación de haz con referencia temporal adaptativo

Como conocemos a priori cuáles serán los símbolos transportados por el conjunto de señales utilizaremos algoritmos lineales para minimizar el error cuadrático medio, MMSE (Minimum Mean Squared Error) que tenemos mediante el entrenamiento de sus pesos en tiempo continuo, ya que está basado en una secuencia de entradas y salidas. Para esto, usaremos dos algoritmos lineales como son el LMS y el RLS.

2.4.1. LMS (Least Mean Square algoritm)

El algoritmo LMS pertenece a la familia de los algoritmos de gradiente estocástico o descenso por gradiente, que utiliza el gradiente para el cálculo de los coeficientes.

Tradicionalmente el filtro lineal adaptativo se centra en la corrección del error mediante el aprendizaje por su capacidad adaptativa pero en el caso del LMS y como vemos en el libro [17], el filtro incorpora un nuevo conjunto de parámetros ajustables: los pesos representados por el vector $\mathbf{w}[n+1]$, un vector con la señal de entrada $\mathbf{x}[n]$ en el instante n y una respuesta estimada $\hat{d}[n]$.

$$\hat{d}[n] = \mathbf{w}^H \mathbf{x}[n] \tag{2.11}$$

La respuesta estimada $\hat{d}[n]$ la comparamos con la respuesta deseada $d[n]$ y se obtiene el siguiente error

$$e[n] = d[n] - \hat{d}[n] = d[n] - \mathbf{w}^H \mathbf{x}[n] \tag{2.12}$$

donde $d[n]$ son las etiquetas deseadas y conocidas a priori y $\mathbf{x}[n]$ es un conjunto de señales de entrenamiento para estas etiquetas deseadas. Debido a esto, este procedimiento se dice que esta supervisado.

Las técnicas de optimización clásicas tienen su origen en la aproximación teórica mediante el algoritmo de descenso por gradiente. Además, asumimos que el error es cuadrático por lo que su función de coste nos quedaría de la siguiente forma:

$$J[n] = E\{\|e[n]\|^2\} \tag{2.13}$$

El error está en función del vector de pesos $\mathbf{w}[n]$ y la idea del algoritmo es modificar este vector para aproximarlo hacia la dirección del descenso por gradiente de $J[n]$, la cual es justamente la opuesta a su gradiente $\nabla_{\mathbf{w}} J[n]$. Asumiendo que las señales son estacionarias y complejas el error es

$$\begin{aligned} E\{\|e[n]\|^2\} &= E\{\|d[n] - \mathbf{w}^H[n]\mathbf{x}[n]\|^2\} \\ &= E\{\|d[n]\|^2 + \mathbf{w}^H[n]\mathbf{x}[n]\mathbf{x}^H[n]\mathbf{w}[n] - 2\mathbf{x}[n]d[n]\} \\ &= \sigma_d^2 + \mathbf{w}^H[n]\mathbf{R}_{\mathbf{xx}}\mathbf{w}[n] - 2\mathbf{p}_{\mathbf{x}d}\mathbf{w}[n] \end{aligned} \tag{2.14}$$

donde $\mathbf{R}_{\mathbf{xx}}$ es la matriz de la señal de autocorrelación, $\mathbf{p}_{\mathbf{x}d}$ es el vector de correlación entre la señal de entrada y la salida del filtro y H es el operador hermítico. Su gradiente con respecto al vector w es expresado como

$$\nabla_{\mathbf{w}} J[n] = 2\mathbf{R}_{\mathbf{xx}}\mathbf{w}[n] - 2\mathbf{p}_{\mathbf{x}}d[n] \tag{2.15}$$

Por lo tanto la regla de adaptación es simple

$$\mathbf{w}[n+1] = \mathbf{w}[n] - \eta\nabla_{\mathbf{w}} J[n] \tag{2.16}$$

El algoritmo LMS fue desarrollado en 1960 por Widrow [18] y es un algoritmo muy sencillo y elegante para el entrenamiento de un sistema lineal mediante la minimización del error cuadrático medio que se aproxima al gradiente $\nabla_{\mathbf{w}} J[n]$ usando una estimación instantánea que escribiremos como

$$\nabla_{\mathbf{w}} J[n] \approx 2\mathbf{x}[n]\mathbf{x}^H[n]\mathbf{w}[n] - 2\mathbf{x}[n]d[n] \tag{2.17}$$

si usamos la aproximación anterior y la aplicamos a la regla de adaptación tenemos la siguiente ecuación

$$\mathbf{w}[n+1] = \mathbf{w}[n] - \eta\mathbf{x}[n]\left(\mathbf{x}^H[n]\mathbf{w}[n] - d[n]\right) \tag{2.18}$$

Simplificamos la regla de adaptación y nos queda finalmente como

$$\mathbf{w}[n+1] = \mathbf{w}[n] - \eta\mathbf{x}[n]e[n] \tag{2.19}$$

2.4.2. RLS (Recursive Least Square algoritm)

Para superar la lentitud de convergencia del LMS se buscó otro algoritmo de filtrado adaptativo, como vemos en el libro [17]. El algoritmo RLS tiene en común con el algoritmo LMS que ambos algoritmos tienen un aprendizaje basado en la corrección del error pero a diferencia del algoritmo LMS, el RLS minimiza además del error cuadrático medio del instante n, todos los errores cuadráticos medios anteriores.

Para empezar, en el algoritmo RLS tenemos una secuencia de datos de entrenamiento de la siguiente forma $\{\mathbf{x}[n], d[n]\}$, y este estima los pesos $\mathbf{w}[n-1]$ minimizando la siguiente función de coste cuadrática

$$min_{\mathbf{w}} \sum_{n=1}^{N-1} \|d[n] - \mathbf{x}^H[n]\mathbf{w}\|^2 \tag{2.20}$$

donde $\mathbf{x}[n]$ es la entrada del regresor de tamaño Lx1 y $d[n]$ es la respuesta deseada, los cuales son denotados de la siguiente forma

$$\begin{aligned} \mathbf{X}[n-1] &= [\mathbf{x}[1], ..., \mathbf{x}[n-1]] \\ d[n-1] &= [d[1], ..., d[n-1]]^H \end{aligned} \tag{2.21}$$

La solución a la función de coste es dada por

$$\mathbf{w}[n-1] = (\mathbf{X}[n-1]\mathbf{X}^H[n-1])^{-1}\mathbf{X}[n-1]d[n-1] \tag{2.22}$$

La minimización de la función de coste se calcula como

$$\mathbf{w}[n] = (\mathbf{X}[n]\mathbf{X}^H[n])^{-1}\mathbf{X}[n]d[n] \tag{2.23}$$

donde definiremos las siguientes matrices para un mejor manejo de las ecuaciones

$$\begin{aligned} \mathbf{P}[n] &= (\mathbf{X}[n]\mathbf{X}^H[n])^{-1} \\ \mathbf{P}[n-1] &= (\mathbf{X}[n-1]\mathbf{X}^H[n-1])^{-1} \end{aligned} \tag{2.24}$$

Utilizando las ecuaciones anteriores, no quedaría la siguiente ecuación resultante

$$\mathbf{P}^{-1}[n] = \mathbf{P}^{-1}[n-1] + \mathbf{x}[n]\mathbf{x}^H[n] \tag{2.25}$$

Para resolver la ecuación tenemos que recurrir a la utilización de la matriz de inversión Lemma, definida como

$$(\mathbf{A} + \mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{C}^{-1} + \mathbf{DA}^{-1}\mathbf{B})^{-1}\mathbf{DA}^{-1} \tag{2.26}$$

cuyos parámetros identificamos y aplicamos con la siguientes asignaciones

$$\mathbf{P}[n-1] \rightarrow \mathbf{A}, \mathbf{x}[n] \rightarrow \mathbf{B}, 1 \rightarrow \mathbf{C}, \mathbf{x}^H[n] \rightarrow \mathbf{D}$$

sustituimos nuestras asignaciones en la ecuación 2.25 y nos da como resultado

$$\mathbf{P}[n] = [\mathbf{P}[n-1] - \frac{\mathbf{P}[n-1]\mathbf{x}[n]\mathbf{x}(n)^H\mathbf{P}[n-1]}{1+\mathbf{x}^H[n]\mathbf{P}[n-1]\mathbf{x}[n]}] \tag{2.27}$$

Con este algoritmo obtenemos la actualización de forma recursiva de los pesos $\mathbf{w}[n]$ directamente de los pesos calculados anteriormente $\mathbf{w}[n-1]$, como vemos a continuación

$$\begin{aligned}\mathbf{w}[n] &= \mathbf{P}[n]\mathbf{X}[n]d[n] = \\ &= (P[n-1] - \frac{\mathbf{P}[n-1]\mathbf{x}[n]\mathbf{x}^H[n]\mathbf{P}[n-1]}{1+\mathbf{x}^H[n]\mathbf{P}[n-1]\mathbf{x}[n]})(\mathbf{x}[n-1]d[n-1] + \mathbf{x}[n]d[n]) = \\ &= \mathbf{w}[n-1] - \frac{\mathbf{P}[n-1]\mathbf{x}[n]\mathbf{x}^H[n]\mathbf{w}[n-1]}{1+\mathbf{x}^H[n]\mathbf{P}[n-1]\mathbf{x}[n]} + \frac{\mathbf{P}[n-1]\mathbf{x}[n]d[n]}{1+\mathbf{x}^H[n]\mathbf{P}[n-1]\mathbf{x}[n]}\end{aligned} \tag{2.28}$$

Simplificando la ecuación anterior tenemos que

$$\begin{aligned}\mathbf{w}[n] &= \mathbf{w}[n-1] + \frac{\mathbf{P}[n-1]\mathbf{x}[n]}{1+\mathbf{x}^H[n]\mathbf{P}[n-1]\mathbf{x}[n]}(d[n] - \mathbf{x}^H[n]\mathbf{w}[n-1]) \\ r[n] &= 1+\mathbf{x}^H[n]\mathbf{P}[n-1]\mathbf{x}[n] \\ \mathbf{k}[n] &= \frac{\mathbf{P}[n-1]\mathbf{x}[n]}{r[n]}\end{aligned} \tag{2.29}$$

donde $\mathbf{k}[n]$ se denomina como el vector de ganancia, y $e[n]$ es la predicción del error. Nótese que $\mathbf{P}[n]$ es de LxL, donde L es el tamaño del vector de entrada $\mathbf{x}[n]$. El algoritmo RLS distribuye la carga computacional de los procesos en cada iteración, conocido en las aplicaciones como canal de ecualización donde los datos están secuencialmente disponibles todo el tiempo.

Una de las principales ventajas que encontramos a priori del RLS con respecto al LMS es que la convergencia del algoritmo RLS es un orden de magnitud más rápida que la del algoritmo LMS, por lo que tardaría menos tiempo en entrenar para obtener el resultado óptimo.

CAPÍTULO 3

PROCESADO DE ARRAYS CON KERNELS

3.1. Kernels

Las máquinas de aprendizaje lineales tienen un coste computacional bajo pero las prestaciones de estas máquinas, en cuestión de probabilidad de error están por debajo del detector óptimo de Bayes. Para acercarnos más a este detector óptimo tenemos que utilizar kernels, como los utilizados por Haykin [15] en su libro.

Aparentemente el problema de designar un filtro adaptativo no lineal es muy difícil, ya que ha habido muchísimos intentos a lo largo del tiempo mediante diferentes sistemas como los modelos de Wiener [8] y Hammerstein [10], pero tenían problemas con la capacidad de estos métodos que era limitada y los sistemas no lineales requerían de una mayor capacidad.

Posteriormente, Gabor [5] intento utilizar las series de Volterra [8] para superar las dificultades matemáticas de los filtros adaptativos no lineales, que aumentan la capacidad de modelado, pero su convergencia es lenta y su complejidad es elevada.

Después surgieron múltiples métodos como son time-laged multilayer perceptrons [4], radial-basis función networks [3] o recurrent neural network [9], entre otros.

Finalmente, se definieron las matrices de Kernel de Mercer [1, 2] una función

continua, simétrica y positiva $\mathbf{k} : \mathbb{U}x\mathbb{U} \rightarrow \mathbb{R}$, donde $\mathbb{U}$ es el dominio de entrada, un subconjunto de $\mathbb{R}^L$. Hay diferentes tipos de kernels, dos ejemplos son el kernel gaussiano y el kernel polinomial y tienen la siguiente forma

- Kernel gaussiano: $\mathbf{k}[\mathbf{x}, \mathbf{x}^H] = \exp(-a||\mathbf{x} - \mathbf{x}^H||^2)$
- Kernel polinomial: $\mathbf{k}[\mathbf{x}, \mathbf{x}^H] = (\mathbf{x} \cdot \mathbf{x}^H)^n$

Las principales propiedades del kernel $\mathbf{k}[\mathbf{x}, \mathbf{x}^H]$, son las que explicaremos a continuación pero para ello suponemos que tenemos dos funciones $\mathbf{h}[\cdot]$ y $\mathbf{g}[\cdot]$ elegidas del espacio de valores reales $\mathbb{H}$ y que son representadas como

$$\begin{aligned} \mathbf{h} &= \sum_{i=1}^{l} \mathbf{a}_i \mathbf{k}[c_i, \cdot] \\ \mathbf{g} &= \sum_{j=1}^{m} \mathbf{b}_j \mathbf{k}[c_j, \cdot] \end{aligned} \tag{3.1}$$

Donde $\mathbf{a}_i$ y $\mathbf{b}_j$ son la expansión de los coeficientes y c_i y c_j pertenecen a $\mathbb{X}$ para todo i y j. La forma bilineal se define como

$$\langle \mathbf{h}, \mathbf{g} \rangle = \sum_{i=1}^{l} \sum_{j=1}^{m} \mathbf{a}_j \mathbf{k}[c_i, c_j] \mathbf{b}_j \tag{3.2}$$

Y que satisface las siguientes propiedades:

1. Simetría

$$\langle \mathbf{h}, \mathbf{g} \rangle = \langle \mathbf{g}, \mathbf{h} \rangle \tag{3.3}$$

2. Escalado y propiedad distributiva

$$\langle (c\mathbf{f} + d\mathbf{g}), \mathbf{h} \rangle = c\langle \mathbf{f}, \mathbf{h} \rangle + d\langle \mathbf{g}, \mathbf{h} \rangle \tag{3.4}$$

3. Norma cuadrática

$$||\mathbf{f}||^2 = \langle \mathbf{f}, \mathbf{f} \rangle \geq 0 \tag{3.5}$$

4. Propiedad de reproducción [1]

$$\begin{aligned} \langle \mathbf{h}, \mathbf{k}[\mathbf{x}, \cdot] \rangle &= \sum_{i=1}^{l} a_j \mathbf{k}[c_i, \mathbf{x}] \\ &= \mathbf{h}[\mathbf{x}] \end{aligned} \tag{3.6}$$

Al espacio $\mathbb{H}$ que cumple con las propiedades anteriores y principalmente con la propiedad de reproducción, se le conoce como Reproducing Kernel Hilbert Space [6] (RKHS).

Un fuerte análisis del espacio RKHS dio lugar a un importante teorema conocido como el teorema de Mercer que establece que todo kernel reproducido $\mathbf{k}[\mathbf{x}, \mathbf{x}^H]$ puede ser expandido como se muestra en la Figura 3.7.

$$\mathbf{k}[\mathbf{x}, \mathbf{x}^H] = \sum_{i=1}^{\infty} \varsigma_i \phi_i[\mathbf{x}] \phi_i[\mathbf{x}^H] \tag{3.7}$$

donde γ_i y ϕ_i son los autovalores y las autofunciones, respectivamente.

Con esto podemos determinar que

$$\mathbf{k}[\mathbf{x}, \mathbf{x}^H] = \phi^H[\mathbf{x}] \phi[\mathbf{x}^H] \tag{3.8}$$

El método de kernel es una herramienta de modelado no paramétrica muy poderosa, y la idea principal es que los datos de entrada son transformados en un espacio de dimensiones altas que pueden ser eficientemente computadas mediante la utilización de kernels. En ese momento es cuando los métodos lineales son apropiados para ser aplicados secuencialmente a los datos transformados, algoritmos de aprendizaje como el LMS y RLS, que aplicados a las matrices de kernel obtenemos los nuevos algoritmos denominados KLMS y KRLS, que definiremos a continuación.

3.2. KLMS (Kernel Least Mean Square)

El algoritmo KLMS es la aplicación del algoritmo lineal LMS en un entorno no lineal transformado en el espacio de Hilbert que posee unas características y propiedades propias, que sirven para adaptar el algoritmo lineal LMS a un entorno no lineal.

Este algoritmo está explicado por Liu en su libro [13] y nos explica que si la asignación entre d y $\mathbf{x}$ es altamente no lineal, poco se puede mejorar con un filtro adaptativo lineal como es el LMS, ya que dicho algoritmo está limitado a los entornos de trabajo lineales. Para adaptar el entorno $\mathbb{U}$ no lineal a otro en el que podamos utilizar dicho filtro lineal, el LMS, tenemos que utilizar las matrices de kernel para transformar los vectores $\mathbf{x}[n]$ en uno dimensionalmente mas alto, $\mathbb{H}$ como $\varphi[\mathbf{x}[n]]$, ya que $\omega^H\varphi[\mathbf{x}]$ es un modelo mucho más poderoso que $\mathbf{w}^H\mathbf{x}$ por la diferencia dimensional entre $\mathbf{x}$ y $\varphi[\mathbf{x}]$.

Por simplicidad utilizaremos

$$\varphi[n] = \varphi[\mathbf{x}[n]] \tag{3.9}$$

Al aplicar el algoritmo LMS al nuevo entorno de kernel, con la secuencia de datos $\varphi[n], d[n]$, tenemos que inicializar los pesos a

$$\omega[0] = 0 \tag{3.10}$$

La actualización de pesos se escribe igual que el LMS lineal y la estimación del error se escribe de la siguiente forma.

$$e[n] = d[n] - \omega^H[n-1]\varphi[n] \tag{3.11}$$

Una vez que tenemos el error en el instante $n-1$, en el instante anterior, procederemos a calcular el peso actual, en el instante n, a partir del peso anterior $\omega[n-1]$, del error cometido y de un parámetros de paso como se muestra en la siguiente ecuación

$$\omega[n] = \omega[n-1] + \eta e[n]\varphi[n] \tag{3.12}$$

donde $\omega[n]$ es la estimación del vector de pesos en $\mathbb{H}$.

Como la dimensión de la matriz de kernel es alta y en el caso del kernel gaussiano es infinita, necesitamos minimizar la carga computacional de ir actualizando

los pesos iteración tras iteración, por lo que intentaremos minimizar la ecuación de actualización de pesos de la siguiente forma.

$$\begin{aligned}\omega[n] &= \omega[n-1] + \eta e[n]\varphi[n] \\ &= (\omega[n-2] + \eta e[n-1]\varphi[n-1]) + \eta e[n]\varphi[n] \\ &= \omega[n-2] + \eta(e[n-1]\varphi[n-1] + e[n]\varphi[n]) \\ &= \dots \\ &= \omega[0] + \eta \sum_{n=1}^{N} e[n]\varphi[n]\end{aligned} \tag{3.13}$$

Si asumimos que $\omega[0] = 0$ podemos definir la ecuación de actualización de pesos como

$$\omega[n] = \eta \sum_{n=1}^{N} e[n]\varphi[n] \tag{3.14}$$

donde la ecuación de estimación de pesos es expresada como una combinación lineal de los datos de entradas y predicción de errores de los instantes actuales y anteriores. Al incluir la ecuación de la actualización de pesos en la ecuación de la salida del filtro adaptativo, tenemos que

$$\begin{aligned}\hat{d}[n-1] &= \omega^H[n]\varphi[n] \\ &= \eta \sum_{i=1}^{n} e[i]\varphi[n]\mathbf{x}^H[n]\varphi[n] \\ &= \eta \sum_{n=1}^{N} e[n](\varphi^H[n]\varphi[n])\end{aligned} \tag{3.15}$$

Aplicamos a esta ecuación el truco del kernel, que consiste en aplicar la siguiente ecuación

$$\mathbf{k}[\mathbf{x}, \mathbf{x}^H] = \varphi^H[n]\varphi[n] \tag{3.16}$$

y nos queda lo siguiente

$$\begin{aligned}\hat{d}[n-1] &= \omega[n]\varphi[n] \\ &= \eta \sum_{n=1}^{N} e[n]\mathbf{k}[\mathbf{x}[n], \mathbf{x}^H[n]]\end{aligned} \tag{3.17}$$

el error estimado seria

$$e[n] = d[n] - \hat{d}[n] \tag{3.18}$$

y la actualización de dicha matriz

$$\hat{d}[n] = \hat{d}[n-1] + \eta e[n]\mathbf{k}[\mathbf{x}[n], \cdot] \tag{3.19}$$

Es importante observar que en esta ecuación ya no actualizamos los pesos, si no que tenemos una suma de todos los errores pasados que se estimaban de las entradas anteriores. Pero ambos procedimientos, tanto con actualización de pesos y la suma de los errores anteriores son equivalentes.

Denominaremos a este nuevo algoritmo KLMS que es el algoritmo LMS en el espacio de Hilbert (RKHS). Este algoritmo trata a los datos de entrada $\mathbf{x}[n]$ como centros y a $\eta e[n]$ como los coeficientes. Pero dichos coeficientes y centros son guardados en memoria durante el entrenamiento. donde a_i es el vector de coeficientes y $\mathbf{x}$ es la entrada que se almacena en memoria.

La salida estimada quedaría como

$$\hat{d}[n] = \eta \sum_{i=1}^{n} e[i]\mathbf{k}[\mathbf{x}[i], \mathbf{x}[i]] \tag{3.20}$$

El KLMS es un algoritmo bastante simple que requiere de $O[n]$ operaciones por filtro. Pero se necesita tener en cuenta varios aspectos para un correcto funcionamiento como son una buena elección del kernel utilizado, del parámetro de paso η y, del crecimiento de la memoria y de la capacidad de computación que se necesitan en operaciones online.

Para una buena elección de η tenemos que tener en cuenta la siguiente ecuación.

$$\eta < \frac{1}{\varsigma_{max}} \tag{3.21}$$

donde ς_{max} es el valor máximo de los autovalores de $\mathbf{R}_\varphi$, el cual es la matriz de autocorrelación de la matriz de los datos de entrada transformados y se calcula como

$$\mathbf{R}_\varphi = \frac{1}{N}\varphi\varphi^H \tag{3.22}$$

donde N es la longitud de la matriz de los datos de entrada.

3.3. KRLS (Kernel Recursive Least Squares)

Este algoritmo [17] deriva del algoritmo lineal RLS aplicado al espacio transformado de kernel de Hilbert (RKHS), en el cuál usamos el teorema de Mercer [1] para transformar los datos de entrada $\mathbf{x}[n]$ en el espacio transformado $\mathbb{H}$ y dichos datos de entrada se transforman en $\varphi[\mathbf{x}[n]$, que nosotros habíamos simplificado por mayor comodidad por $\varphi[n]$.

Definimos las secuencias de entrenamiento como $\{d[1],d[2],...\}$ y $\{\varphi[1],\varphi[2],...\}$. Para calcular el vector de pesos $\omega[n]$ en cada iteración tenemos que minimizar la siguiente ecuación

$$min_{\omega}\sum_{i=1}^{n}|d(i)-\omega^{H}\varphi[i]|^{2}+\lambda\|\omega\|^{2} \tag{3.23}$$

donde $\lambda\|\omega\|^2$ es el termino de penalización de la norma o parámetro de regularización, al minimizar la función anterior, obtenemos como resultado que

$$\omega[n]=(\lambda I+\varphi[n]\varphi^{H}[n])^{-1}\varphi[n]d[n] \tag{3.24}$$

y para resolverlo usaremos la matriz de inversión Lemma que simplifica la ecuación 3.24

$$(\mathbf{A}+\mathbf{BCD})^{-1}=\mathbf{A}^{-1}-\mathbf{A}^{-1}\mathbf{B}(\mathbf{C}-1+\mathbf{DA}^{-1}\mathbf{B})^{-1}\mathbf{DA}^{-1} \tag{3.25}$$

realizamos las asignaciones de los parámetros

$$\lambda\mathbf{I}\rightarrow\mathbf{A},\varphi[n]\rightarrow\mathbf{B},\mathbf{I}\rightarrow\mathbf{C},\varphi^{H}[n]\rightarrow\mathbf{D}$$

Sustituimos las asignaciones a la ecuación y obtenemos que

$$(\lambda\mathbf{I}+\varphi[n]\varphi^{H}[n])^{-1}\varphi[n]=\varphi[n](\lambda\mathbf{I}+\varphi^{H}[n]\varphi[n])^{-1} \tag{3.26}$$

sustituyendo en la ecuación 3.24 nos da como resultado

$$\omega[n]=\varphi[n](\lambda\mathbf{I}+\varphi^{H}[n]\varphi[n])d[n] \tag{3.27}$$

En esta ecuación podemos observar varias cosas. La primera, es que mediante el truco de kernel podemos sustituirlo en la ecuación 3.27.

$$\mathbf{k}[\mathbf{x},\mathbf{x}^{H}]=\varphi^{H}[n]\varphi[n] \tag{3.28}$$

y la segunda cosa que podemos observar es que los pesos se pueden expresar como una combinación lineal de los datos de entrada de la siguiente forma

$$\omega[n] = \varphi[n]\mathbf{a}[n] \tag{3.29}$$

donde

$$\mathbf{a}[n] = (\lambda\mathbf{I} + \varphi^H[n]\varphi[n])^{-1}d[n] \tag{3.30}$$

Ahora definiremos la matriz $\mathbf{Q}$,

$$\mathbf{Q}[n] = (\lambda\mathbf{I} + \varphi^H[n]\varphi[n])^{-1} \tag{3.31}$$

hacemos la inversa de la matriz $\mathbf{Q}[n]$ y nos quedaría como la siguiente ecuación

$$\mathbf{Q}^{-1}[n] = \begin{bmatrix} \mathbf{Q}^{-1}[n-1] & h[n] \\ h^H[n] & \lambda + \varphi^H[n]\varphi[n] \end{bmatrix} \tag{3.32}$$

donde

$$h[n] = \varphi^H[n]\varphi[n] \tag{3.33}$$

Como utilizamos la estructura de ventana deslizante para actualizar la matriz de crecimiento inversa, que mostramos a continuación

$$\mathbf{Q}[n] = r^{-1}[n]\begin{bmatrix} \mathbf{Q}[n-1]r[n] + z[n]z^H[n] & -z[n] \\ -z^H[n] & 1 \end{bmatrix} \tag{3.34}$$

donde

$$\begin{aligned} z[n] &= \mathbf{Q}[n-1]h[n] \\ r[n] &= \lambda + \varphi^H[n]\varphi[n] - z^H[n]h[n] \end{aligned} \tag{3.35}$$

Por otro lado, la expansión de coeficientes de los pesos son

$$\begin{aligned} \mathbf{a}[n] &= \mathbf{Q}[n]d[n] = \\ &= \begin{bmatrix} \mathbf{Q}[n-1] + z[n]z^H[n]r^{-1}[n] & -z[n]r^{-1}[n] \\ -z^H[n]r^{-1}[n] & r^{-1}[n] \end{bmatrix}\begin{bmatrix} d[n-1] \\ d[n] \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{a}[n-1] - z[n]r^{-1}[n]e[n] \\ r^{-1}[n]e[n] \end{bmatrix} \end{aligned} \tag{3.36}$$

donde $e[n]$ es la predicción del error de los datos ya computados, siendo esté la diferencia entre la señal o símbolos deseados y la predicción de la señal $y[n-1]$:

$$\begin{aligned} \hat{d}[n-1] &= h^H[n]\mathbf{a}[n-1] \\ &= \sum_{i=1}^{n-1} \mathbf{a}_i[n-1]\mathbf{k}[\mathbf{x}[i], \mathbf{x}[n]] \end{aligned} \tag{3.37}$$

$$e[n] = d[n] - \hat{d}[n-1] \tag{3.38}$$

Podemos observar que el procedimiento de aprendizaje del KRLS es similar al del KLMS, en el sentido de que asociamos las nuevas entradas al algoritmos como centros y que también podemos usar el error estimado como si fueran los pesos. Pero al contrario que en el KLMS, en el KRLS no solo actualizamos el peso actual si no que también actualizamos los pesos anteriores.

También, al igual que en el KLMS almacenamos los coeficientes $\mathbf{a}[n]$ y los centros en la memoria durante el entrenamiento lo que no hace tener una mayor carga computacional y uso de memoria para almacenar dichos datos.

El algoritmo KRLS tiene una complejidad en espacio y tiempo del orden de $O[n^2]$. Terminaremos simplificando la última ecuación de la salida estimada nos daría el siguiente resultado.

$$\hat{d}[n] = \sum_{i=1}^{n} \mathbf{a}_i[n]\mathbf{k}[\mathbf{x}[i], \mathbf{x}[\mathbf{n}]] \tag{3.39}$$

CAPÍTULO 4

PROCESADO CON KERNEL RECURSIVO

En el pasado el procesado de datos no lineales mediante métodos de kernel creció su popularidad debido al buen equilibrio que se observa entre la mejora, la estabilidad y la robustez de los algoritmos. Pero su potencial para ser aplicado a escenarios adaptativos reales estaba restringido debido a que cada vez que teníamos un nuevo parámetro, había que volver a reentrenar los pesos para seguir minimizando nuestro error. Por lo que se optó por la utilización de métodos recursivos.

Nuestro método de kernel recursivo del articulo [16] y que desarrollaremos consta principalmente de 4 fases: la fase de recepción de datos, la fase de crecimiento, la fase de poda y la fase de aplicación de algoritmo.

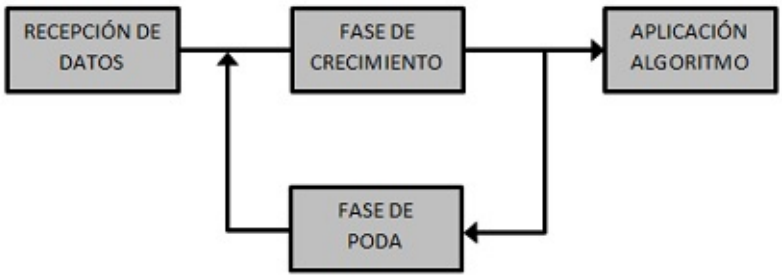

Figura 4.1: Sistema de kernel recursivo

4.1. Fase de recepción de datos

En esta primera fase lo que hacemos es recibir los datos de entrada y transformarlos al espacio de kernel reproducido de Hilbert (RKHS). Una vez transformados, los datos son enviados a la fase de crecimiento donde serán introducidos en la matriz principal P=1 y se generarán los datos que introduciremos en las P-1 matrices recursivas.

4.2. Fase de crecimiento

Nuestra matriz de kernel recursiva consta como veremos más adelante, de una matriz con los datos de kernel y de P-1 matrices de kernel, las cuales contienen residuos de los datos de kernel anteriores. Por eso cuando actualizamos la matriz de kernel recursiva, no solamente tenemos que actualizar la matriz principal, sino que también deberemos actualizar las matrices de residuales con los nuevos datos recursivos.

Estos datos residuales de los datos de kernel anteriores son importantes porque serán utilizados en nuestros algoritmos de aprendizaje, en el cálculo del error estimado y en la estimación de sus pesos, y por lo tanto ayudarán a la estabilidad de nuestro sistema.

Cuando recibimos los datos transformados, los añadimos a los que ya teníamos de la matriz de kernel principal, la cual actualizamos y también actualizamos nuestras matrices de kernel P-1 recursivas.

A continuación, veremos cómo se ha desarrollado el método que utilizamos para la matriz recursiva y la salida de dicha matriz que usaremos en los algoritmos de aprendizaje.

4.2.1. Matriz Recursiva de Kernel

Aquí se propondrá un método alternativo al de utilizar los algoritmos de aprendizaje directamente en las matrices de kernel y se define como un modelo de recursión explícito de la señal en el RKHS. La metodología para la obtención de dicha matriz recursiva sigue tres pasos. El primer paso es definir un modelo recursivo que este estructurado directamente en RKHS. El segundo es explotar el teorema adecuado para representar los pesos para el modelo de aprendizaje genérico. Y el tercer paso y último es aplicar las formulas generales de recursión para el procesado clásico de la señal en RKHS.

Los kernels obtenidos e implementados en RKHS pueden ser expresados como una función de kernels previamente computados y el modelado de sus parámetros mantengan su completo significado. Actualmente, un filtro definido apropiadamente en RKHS nos permite desarrollar un kernel adaptativo eficiente.

Se quiere tener el control de la profundidad de la memoria (M_H) en RKHS, por lo que se implementa el filtro gamma [11] recursivo en RKHS mediante las estructura de los filtros FIR, donde tendremos que $M_H = P/\mu$ con $\mu \in [0,1]$. Nuestro objetivo se centra en la formalización de la recursión en el espacio de kernel y así, aplicar las configuraciones adaptativas del kernel recursivo.

Ahora adaptaremos el estándar del filtro gamma [11] al entorno RKHS, y el estándar del filtro gamma es definido con la siguiente expresión

$$
\begin{aligned}
y[n] &= \sum_{i=i}^{P} \mathbf{w}_i \mathbf{x}^i[n] \\
\mathbf{x}^i[n] &= \begin{cases} \mathbf{x}[n], & i = 1 \\ (1-\mu)\mathbf{x}^i[n-1] + \mu \mathbf{x}^{i-1}[n-1] & 2 \leq i \leq P \end{cases}
\end{aligned}
\tag{4.1}
$$

donde $y[n]$ es la salida del filtro, $\mathbf{x}[n]$ es la entrada al filtro, y P y μ son los parámetros que controlan la estabilidad y la profundidad de la memoria.

Transformamos las formulas anteriores para que estén expresadas en el espacio de Hilbert, y tiene la siguiente forma

$$\begin{aligned} y[n] &= \sum_{i=i}^{P} \omega_i \phi^i[n] \\ \phi^i[n] &= \begin{cases} \varphi[n], & i = 1 \\ (1-\mu)\varphi^i[n-1] + \mu\varphi^{i-1}[n-1] & 2 \leq i \leq P \end{cases} \end{aligned} \tag{4.2}$$

donde $\varphi[\cdot]$ es la transformación no lineal en RKHS y $\phi^i[n]$ no debe ser la imagen de un vector que pertenezca al espacio de entrada, $\phi^i[n] \neq \varphi[\mathbf{x}^i[n]]$.

Podemos observar en la primera fórmula, la de la salida del filtro, que los vectores de pesos $\mathbf{w}_i$ al ser lineales pueden ser formulados como

$$\mathbf{w}_i = \sum_{m=1}^{N} \beta_m^i \phi_m^i \tag{4.3}$$

y aplicando a la ecuación 4.3 el truco del kernel,

$$\mathbf{K}^i[m,n] = \langle \phi_m^i \phi_n^i \rangle \tag{4.4}$$

obtenemos que la salida del filtro queda como

$$\begin{aligned} y[n] &= \sum_{i=1}^{P} \sum_{m=1}^{N} \beta_m^i \phi^i[m] \phi^i[n] \\ &= \sum_{i=1}^{P} \sum_{m=1}^{N} \beta_m^i \mathbf{K}^i[m,n] \end{aligned} \tag{4.5}$$

Ahora transformaremos la segunda ecuación $\phi^i[n]$ utilizando el truco del kernel

$$\mathbf{K}^i[m,n] = \langle \phi^i[n] \phi^i[n] \rangle \tag{4.6}$$

y dando como resultado la siguiente ecuación

$$\mathbf{K}^i[m,n] = \begin{cases} \mathbf{K}[\mathbf{x}[m], \mathbf{x}[n]], & i = 1 \\ \langle (1-\mu)\phi^i[m-1] + \mu\phi^{i-1}[m-1], (1-\mu)\phi^i[n-1] + \\ \mu\phi^{i-1}[n-1] \rangle & 2 \leq i \leq P \end{cases} \tag{4.7}$$

Si ordenamos los términos y los agrupamos, obtenemos que

$$\mathbf{K}^i[m,n] = \begin{cases} \mathbf{K}[x[m],x[n]], & i=1 \\ (1-\mu)^2\mathbf{K}^i[m-1,n-1]+\mu^2\mathbf{K}^{i-1}[m-1,n-1] & \\ +\mu(1-\mu)\langle\phi^i[m-1],\phi^{i-1}[n-1]\rangle & \\ +\mu(1-\mu)\langle\phi^{i-1}[m-1],\phi^i[n-1]\rangle & 2\leq i\leq P \end{cases} \tag{4.8}$$

Pero todavía no hemos podido agruparlo del todo y lo que vamos a hacer es aplicar la recursión otra vez sobre $\langle\phi^i[m-1],\phi^{i-1}[n-1]\rangle$ que es muy parecido a $\langle\phi^{i-1}[m-1],\phi^i[n-1]\rangle$, solo hay que cambiar parámetros y nos sale

$$\langle\phi^i[m-1],\phi^{i-1}[n-1]\rangle = \begin{cases} 0, & i=1 \\ \langle(1-\mu)\phi^i[m-2] & \\ +\mu\phi^{i-1}[m-2],\phi^{i-1}[n-1]\rangle & 2\leq i\leq P \end{cases} \tag{4.9}$$

reagrupando los parámetros de nuevo, obtenemos que

$$\langle\phi^i[m-1],\phi^{i-1}[n-1]\rangle = \begin{cases} 0, & i=1 \\ (1-\mu)\langle\phi^i[m-2],\phi^{i-1}[n-1]\rangle & \\ +\mu\mathbf{K}^{i-1}[m-2,n-1] & 2\leq i\leq P \end{cases} \tag{4.10}$$

Vemos que nos sigue quedando dependiendo de $\langle\phi^i[m-2],\phi^{i-1}[n-1]\rangle$, que es parecido al que queremos agrupar pero utilizando los datos de la matriz de instantes anteriores. Por lo que se está usando de forma recursiva junto a $\mathbf{K}^{i-1}[m-2,n-1]$ sobre los datos que ya hemos calculado. Por lo que asumiendo que es recursivo y que $\phi^i[n]=0$ para $n<0$, podemos agrupar los parámetros recursivos y nos quedaría la siguiente ecuación

$$\langle\phi^i[m-1],\phi^{i-1}[n-1]\rangle = \begin{cases} 0, & i=1 \\ \mu\sum_{j=2}^{m}(1-\mu)^{j-2}\mathbf{K}^{i-1}[m-j,n-1] & 2\leq i\leq P \end{cases} \tag{4.11}$$

Finalmente el kernel recursivo quedaría como

$$\mathbf{K}^i[m,n] = \begin{cases} \mathbf{K}[\mathbf{x}[m],\mathbf{x}[m]], & i=1 \\ (1-\mu)^2\mathbf{K}^i[m-1,n-1]+\mu^2\mathbf{K}^{i-1}[m-1,n-1] & \\ +\mu^2\sum_{j=2}^{m-1}(1-\mu)^{j-1}(\mathbf{K}^{i-1}[m-j,n-1] & \\ +\mathbf{K}^{i-1}[m-1,n-j]) & 2\leq i\leq P \end{cases} \tag{4.12}$$

Como vemos este kernel está construido de forma recursiva y el parámetro μ conserva sus características para el control de la profundidad de la memoria en función del espacio, a través de las siguientes propiedades

1. El kernel del filtro gamma [11] recursivo es definido y positivo.
2. El filtro gamma [11] en el kernel recursivo en la ecuación anterior es una reproducción del kernel en el espacio $\mathbb{H}$.
3. El conjunto de características de Aronszajn [1] que corresponden al filtro de kernel recursivo gamma [11] es único.
4. El filtro gamma [11] recursivo generaliza el filtrado recursivo de los filtros FIR en RKHS.
5. El filtro recursivo de kernel gamma generaliza para los filtros gamma [11].
6. El filtro de kernel recursivo gamma [11] generaliza los filtros recursivos AR.
7. los modelos ARMA [14] pueden ser obtenidos a través de una combinación linear sobre modelos recursivos individuales.

4.3. Fase de Aplicación del Algoritmo

En esta fase, se utilizarán los algoritmos de kernel seleccionados el KLMS y el KRLS, en nuestro caso haremos la modificación para el KLMS y calculamos la probabilidad de error de la siguiente forma

$$\begin{aligned} e[n] &= d[n] - \hat{d}[n] \\ &= d[n] - \omega^H[n-1]\varphi[n] \end{aligned} \tag{4.13}$$

no se utilizara dicha ecuación ya que la salida de la matriz de kernel $\hat{d}[n]$, será diferente y en su lugar, en la matriz de kernel recursivo se utiliza la siguiente

$$\hat{d}[n] = \sum_{i=1}^{P} \omega^H[n-1]\mathbf{K}^i[:,n] \tag{4.14}$$

Por lo que la estimación del error también será diferente

$$\begin{aligned} e[n] &= d[n] - \hat{d}[n] \\ &= d[n] - \sum_{i=1}^{P} \omega^H[n-1]\mathbf{K}^i[:,n] \end{aligned} \tag{4.15}$$

Y se obtendrán diferentes valores de los pesos estimados por lo que la salida final del sistema será diferente, al igual que la probabilidad de error y la estabilidad del sistema también cambiaran.

4.4. Fase de Poda

Esta fase es creada principalmente para que la matriz no tenga un crecimiento ilimitado y pueda utilizarse de forma recursiva. Para ello, lo que se hará es fijar un tamaño fijo de la matriz de kernel y está fase solamente empezara a aplicarse el método de poda cuando el tamaño de dicha matriz se haya alcanzado.

El método de poda que se utilizara aquí es un método muy sencillo y consiste básicamente en el descarte de los datos almacenados en la matriz más antiguos, tanto en la matriz principal de kernel recursivo como en las matrices $P-1$ que contienen los residuos de la recursión.

A la hora de eliminar los datos más antiguos en nuestras matrices de kernel consiste en eliminar la primera fila y la primera columna de todas las matrices, quedando la matriz preparada para añadir nuevos datos de entrada.

CAPÍTULO 5

EXPERIMENTOS

5.1. Introducción

En este capítulo de experimentos lo que vamos a hacer es aplicar la matriz de kernel recursivo que hemos explicado y desarrollado en el capítulo anterior, junto con los algoritmos lineales que también se han explicado, a los datos que generaría una antena de array. Está matriz de kernel recursiva minimizará el error cometido en la recepción de la transmisión de señales por parte de los usuarios.

Los resultados que vamos a mostrar son el comportamiento con el transcurso del tiempo de está matriz recursiva y cómo se comporta y varía su probabilidad de error de bit (BER) cuando variamos diferentes parámetros de la matriz de kernel, la SNR y el escenario en el que se encuentra.

5.2. Escenarios

En nuestros experimentos sólo trabajemos en dos tipos de escenarios: un escenario estacionario y un escenario no estacionario.

En ambos escenarios, se trabajará con una antena del articulo [19], la cual es una antena de array de antenas de 7 elementos, estos elementos son del tipo dipolo asimétrico grueso y están separados entre sí, una distancia de $\lambda/4$. Dicha antena trabaja a una frecuencia de 3 GHz y obtiene una frecuencia de símbolo de 10 MHz. Cada trama genera 26 símbolos de datos de entrenamiento y 146 símbolos de datos de test, con sus correspondientes etiquetas de entrenamiento y de test.

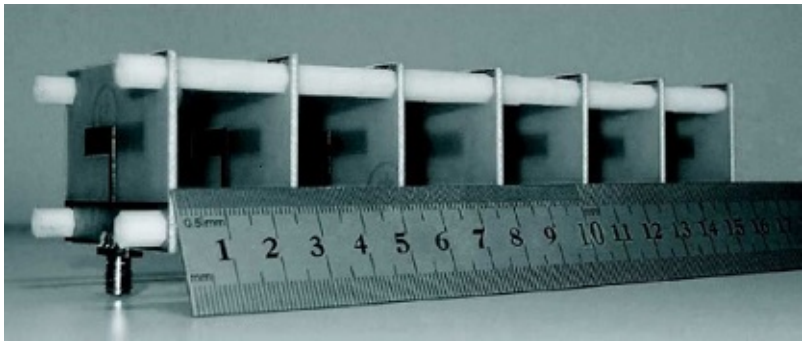

Figura 5.1: Antena de array de 7 elementos separados entre sí $\lambda/4$, a una frecuencia de 3 GHz y una frecuencia de símbolo de 10 MHz.

También es común en ambos escenarios el que existen 4 usuarios transmitiendo al mismo tiempo con una modulación de BPSK, con una relación señal ruido (SNR) que varía desde 0dB hasta 22dB y un ruido del tipo gaussiano blanco. Estos usuarios a los que asignaremos los números 1, 2, 3 y 4 poseen un ángulo de llegada de -10º, 0º, 10º y 20º respectivamente.

La diferencia entre cada escenario es la siguiente:

En el escenario estacionario, la amplitud del usuario del que queremos recibir la señal es estable y no varía, por lo que tendrá el valor de A=1 para todas las tramas generadas desde la primera trama hasta la última.

En cambio para el escenario no estacionario la amplitud de las señales no es constante y estable cómo se da en el caso anterior, si no que varía entre un intervalo de tramas. Por lo que tenemos que las señales empiezan a transmitir de forma constante y con una amplitud máxima de A=1 hasta la trama 65. A partir de la trama 66 hasta la trama 130, la amplitud de las señales disminuye un 20 % y toma el valor de A=0.8. Pero a partir de la trama 131 las señales recuperan otra vez su amplitud máxima de A=1 hasta el final.

5.3. Experimentos a realizar

En esta sección, definiremos y explicaremos los diferentes experimentos que vamos a realizar utilizando la matriz de kernel recursiva y observaremos si se obtiene un mejor comportamiento de esta matriz al compararla con la matriz de kernel no recursiva.

5.3.1. Experimento 1

En este primer experimento, utilizaremos junto a la matriz de kernel recursiva el algoritmo KLMS y compararemos el comportamiento de la matriz recursiva y de la no recursiva y observaremos si se produce mejoría al usar la matriz de kernel recursiva.

Realizaremos las simulaciones para los dos casos de la matriz de kernel, el no recursivo (KLMS) y el recursivo (RKHS-KLMS). Se generará 20 tramas en un entorno estacionario con una SNR de 10dB y donde el usuario deseado es el 4 para cada caso, por lo que la amplitud de las señales de cada usuarios son constantes e igual a 1. Y cada simulación se repetirá 20 veces para obtener los resultados promediados.

Al usar el algoritmo de aprendizaje KLMS, y cómo este algoritmo tienen varios parámetros libres, tenemos que fijarlos a los siguientes valores: para el parámetro de paso, elegimos $\eta = 0{,}001$ y utilizaremos el kernel de tipo gaussiano en el que el parámetro de kernel es 0,1.

La matriz de kernel recursivo también tiene varios parámetros que se tienen que fijar a un determinado valor para la simulación, como son el tamaño de la matriz de kernel que tomará el valor de 200 símbolos, el parámetro de profundidad de la memoria P=5 y el parámetro de control de memoria $\mu = 0{,}9$.

Con los datos obtenidos de este experimento representaremos las curvas de aprendizaje de los dos casos, las compararemos y observaremos su posible mejoría al utilizar el caso recursivo.

5.3.2. Experimento 2

En este segundo experimento queremos observar si se produce mejoría entre el caso recursivo y el no recursivo al comparar las probabilidades de error de bit de cada caso con el transcurso de las tramas.

Generaremos 20 tramas en un entorno estacionario con una SNR de 10dB y donde el usuario deseado es el 4 para realizar las simulaciones, y tenemos que tener en cuenta que es un entorno estacionario por lo que las señales de cada usuario no varían y son iguales a 1. Además, cada simulación la repetiremos 20 veces para obtener unos resultados promediados y que no tengan valores atípicos.

Utilizamos el algoritmo de aprendizaje KLMS, y para poder utilizarlo tenemos que fijar sus parámetros libres a los siguientes valores: para el parámetro de paso, elegimos $\eta = 0{,}001$ y utilizaremos el kernel gaussiano en el que el parámetro de kernel es 0,1.

Ahora ajustaremos los valores de la matriz de kernel, en la cual, se usará un tamaño de matriz de kernel de 200 símbolos, un parámetro de profundidad de memoria de P=5 y un control de dicha profundidad de $\mu = 0{,}9$.

De los resultados obtenidos, representaremos las gráficas de las probabilidades de error de bit de cada simulación para caso no recursivo (KLMS) y el caso recursivo (RKHS-KLMS). Comprobando si el caso recursivo obtiene una probabilidad de error de bit inferior que la del caso no recursivo.

5.3.3. Experimento 3

En el tercer experimento lo que queremos ver es cómo se comporta el algoritmo KLMS con la matriz de kernel cuando variamos el tamaño de la matriz de kernel y observaremos cómo evolucionan para cada tamaño de la matriz con el paso de las tramas para ambos casos, el caso recursivo (RKHS-KLMS) y el no recursivo (KLMS).

Para el experimento se generarán 20 tramas en un entorno estacionario con una SNR de 10dB y donde el usuario deseado es el 4 para cada caso, y sabemos que las amplitudes de la señal de cada usuario son constante e igual a 1 para todas las tramas. Dicha simulación se repetirá 20 veces para la obtención de unos resultados promediados que no tienen valores atípicos.

Ahora asignaremos valores a los parámetros libres que utiliza el algoritmo de aprendizaje KLMS para su funcionamiento, para el parámetro de paso elegimos $\eta = 0{,}001$ y utilizaremos el kernel gaussiano en el que el parámetro de kernel es 0,1.

En la matriz de kernel, fijaremos el valor de los parámetros de profundidad de memoria P=5 y del control de la profundidad $\mu = 0{,}9$. Pero el valor del tamaño de la matriz de kernel recursiva será de 100 símbolos, 150 símbolos, 200 símbolos, 250 símbolos y de 300 símbolos para ambos casos.

Realizaremos 5 gráficas, una para cada tamaño de la matriz de kernel, en la que se representaran las probabilidades de error de bit de cada caso en función de las tramas y, veremos qué caso se comporta mejor para cada valor del tamaño de la matriz y cómo evoluciona la probabilidad de error de bit al aumentar el tamaño de la matriz de kernel.

5.3.4. Experimento 4

En el cuarto experimento lo que queremos comprobar es cómo se comporta el algoritmo KLMS con la matriz de kernel recursivo cuando le variamos otro parámetro de esta matriz, esta vez vamos a variar el parámetro de profundidad de la memoria, P. Dicho valor variará desde P=2 a P=5 y veremos cómo se minimizará la probabilidad de error de bit para cada valor de P y si mejora o empeora dicha probabilidad en el caso de utilizar el caso recursivo (RKHS-KLMS) o el no recursivo (KLMS).

Se generaran 20 tramas en un entorno estacionario con una SNR de 10dB y donde el usuario deseado es el 4 para la simulación de cada caso. Al ser un escenario estacionario la amplitud de todas las señales de cada usuario es constante e igual a 1. Además, repetiremos las simulaciones 20 veces para la obtención de unos resultados promediados y sin valores atípicos.

Al usar el algoritmos de aprendizaje KLMS, tenemos que fijar sus parámetros libres que tomarán los mismos valores que en los anteriores experimentos: para el parámetro de paso elegimos $\eta = 0{,}001$ y utilizaremos el kernel de tipo gaussiano en el que el parámetro de kernel es 0,1.

Los parámetros de la matriz de kernel tomarán los siguientes valores: en este caso el valor del tamaño de la matriz será fijo a 200 símbolos y el control de la profundidad será de $\mu = 0{,}9$. Pero en este experimento variaremos el parámetro P que tomara los valores de P=2, P=3, P=4 y P=5, para ambos casos.

Los resultados obtenidos de las simulaciones serán representados en 4 gráficas, una para cada valor de P, en las que se mostrará la evolución de la probabilidad de error de bit con el paso de las tramas. Y veremos si probabilidad de error obtenida al usar la matriz de kernel recursiva se comporta mejor que el no recursivo para los valores de P y como varía esta probabilidad de error al aumentar el valor del parámetro P.

5.3.5. Experimento 5

En este experimento no vamos a variar ningún parámetro de la matriz de kernel recursiva, si no que variaremos la relación señal ruido al generar los datos de recepción de la antena, variaremos la SNR, que tomará valores entre 4dB y 22dB, y veremos como varia la probabilidad de error de bit cuando aumentamos la SNR para ambos casos, el caso no recursivo (KLMS) y el recursivo (RKHS-KLMS).

Para obtener los datos que simularemos, generaremos 20 tramas en un entorno estacionario con una SNR que variaremos y donde el usuario deseado es el 4. El parámetro SNR tomará los valores de 4dB, 7dB, 10dB, 13dB, 16dB, 19dB y 22dB. Luego los usaremos para realizar las simulaciones para el caso no recursivo y el caso recursivo, y los repetiremos 20 veces para que los resultados de las simulaciones estén promediados y no contengan valores atípicos.

Para usar el algoritmo de aprendizaje KLMS tenemos que fijar su parámetros libres que tomarán los siguientes valores: para el parámetro de paso elegimos $\eta = 0{,}001$ y utilizaremos el kernel gaussiano en el que el parámetro de kernel es 0,1.

En este experimento fijaremos todos los parámetros de la matriz de kernel recursiva, ya que los que vamos a variar es la SNR que genera los datos que después simularemos. El tamaño de la matriz de kernel será de 200 símbolos, el parámetro de profundidad de la memoria P=5 y el parámetro de control de la profundidad $\mu = 0{,}9$.

Los resultados obtenidos de las simulaciones las representaremos en una gráfica donde compararemos cómo evolucionan las probabilidades de error de bit cuando aumentamos la SNR para el caso no recursivo y el caso recursivo, y veremos si se obtendrá una probabilidad de error de bit menor con el caso recursivo. Y también veremos cómo se comportan las probabilidades de error de bit en cada caso al aumentar el valor de la SNR.

5.3.6. Experimento 6

En este penúltimo experimento, lo que vamos a comprobar es el comportamiento que obtenemos con la matriz de kernel recursivo junto con el algoritmo de aprendizaje KLMS en un entorno no estacionario comparándola con el comportamiento obtenido al no usar la matriz de kernel recursiva. Y podremos ver si conseguimos que dicha matriz sea más estable que la no recursiva.

En este caso, generaremos 200 tramas en un entorno no estacionario para comprobar mejor la estabilidad con el paso de las tramas. Ya que cómo mencionamos en la sección de los tipos de escenarios, la amplitud de las señales de los usuarios tendrán un valor de 0.8 para tramas que van desde la 66 hasta la 130, para el resto de tramas toma el valor de 1. Por lo que observaremos cómo se comporta el sistema con la perdida de potencia de las señales emitidas por los usuarios. Tomaremos un valor de la SNR de 10dB y donde el usuario deseado es el 4 para la simulación de cada caso. Y repetiremos cada simulación 20 veces para obtener los resultados promediados y que no contengan valores atípicos.

Ahora fijamos los parámetros libres del algoritmo KLMS, done elegimos el parámetro de paso a $\eta = 0{,}001$ y utilizaremos el kernel gaussiano en el que el parámetro de kernel es 0,1.

En este experimento fijamos todos los parámetros de la matriz de kernel recursiva ya que solo queremos ver la estabilidad de este algoritmo. Los parámetros serán los siguientes: el tamaño de la matriz será de 200 símbolos, con una profundidad de memoria de P=5 y un control de la profundidad de $\mu = 0{,}9$.

Los resultados obtenidos de las simulaciones realizadas las representaremos en una gráfica donde observaremos cómo se comportan la matriz de kernel recursiva y la no recursiva al producirse una caída de la potencia de las señales transmitidas en determinadas tramas, y veremos si el caso recursivo minimiza mejor la probabilidad de error de bit que el caso no recursivo con el paso de las tramas.

5.3.7. Experimento 7

En el último experimento haremos una comparación de diferentes algoritmos que se han obtenido de un ejemplo del artículo publicado [16], lo que haremos es representar dicha gráfica y comentar los resultados observados.

Sabemos del artículo [16] que se generaron 200 tramas en un entorno no estacionario con una SNR de 10dB y el usuario deseado es el 4. Los datos simulados fueron repetidos 10 veces.

Sabemos también que se fijaron los parámetros libres para algoritmo KLMS, el parámetro de paso es $\eta = 0{,}001$ y el KRLS tiene un factor de olvido de 0,99 y un parámetro de regularización de 0,001. Y se fijó el parámetro de la matriz de kernel $\mu = 0{,}9$.

5.4. Experimento 1

Representaremos los resultados del experimento comparando el caso no recursivo (KLMS) con el caso recursivo (RKHS-KLMS) mediante su curva de aprendizaje, en la Figura 5.2.

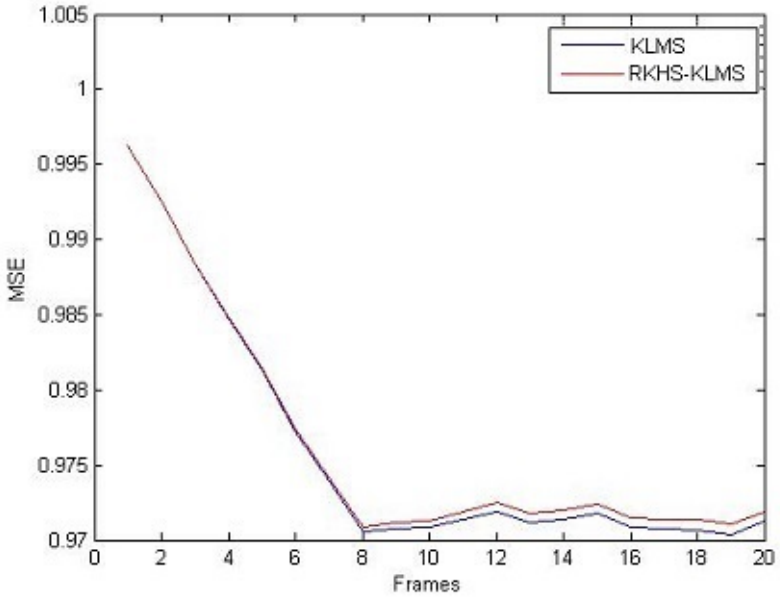

Figura 5.2: Comparación de las curvas de aprendizaje del KLMS y RKHS-KLMS en una gráfica de 20 tramas, con una $\eta = 0{,}001$, una SNR de 10dB, un tamaño de matriz de kernel de 200 símbolos, una profundidad de memoria de P=5 y un control de dicha profundidad de $\mu = 0{,}9$.

En la gráfica anterior, vemos las curvas de aprendizaje del KLMS no recursivo y del KLMS recursivo y vemos que es mejor la del KLMS no recursivo, ya que desciende un poco más que el caso recursivo. Por lo que no habrá mejora en este experimento.

5.5. Experimento 2

Representaremos los resultados del experimento, comparando el caso no recursivo (KLMS) con el caso recursivo (RKHS-KLMS) mediante sus probabilidades de error con el paso de las tramas, en la Figura 5.3.

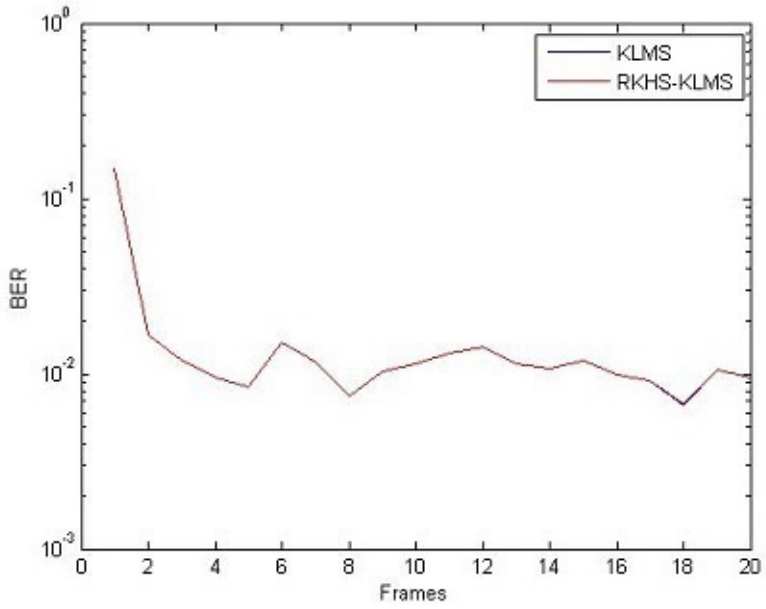

Figura 5.3: Comparación de las probabilidades de error de bit, BER, del KLMS y RKHS-KLMS en una gráfica de 20 tramas, con una $\eta = 0{,}001$, una SNR de 10dB, un tamaño de matriz de kernel de 200 símbolos, una profundidad de memoria de P=5 y un control de dicha profundidad de $\mu = 0{,}9$.

En la gráfica anterior podemos observar que no se aprecia ninguna mejoría del caso recursivo con respecto al caso no recursivo, ya que las probabilidades de error con el paso de las tramas son casi idénticas.

5.6. Experimento 3

Representaremos los resultados del experimento, comparando el caso no recursivo (KLMS) con el caso recursivo (RKHS-KLMS) mediante sus probabilidades de error con el paso de las tramas, las tenemos que hacer para cada valor del tamaño de la matriz de kernel. Empezaremos por el de tamaño igual a 100 símbolos que ser representará en la Figura 5.4.

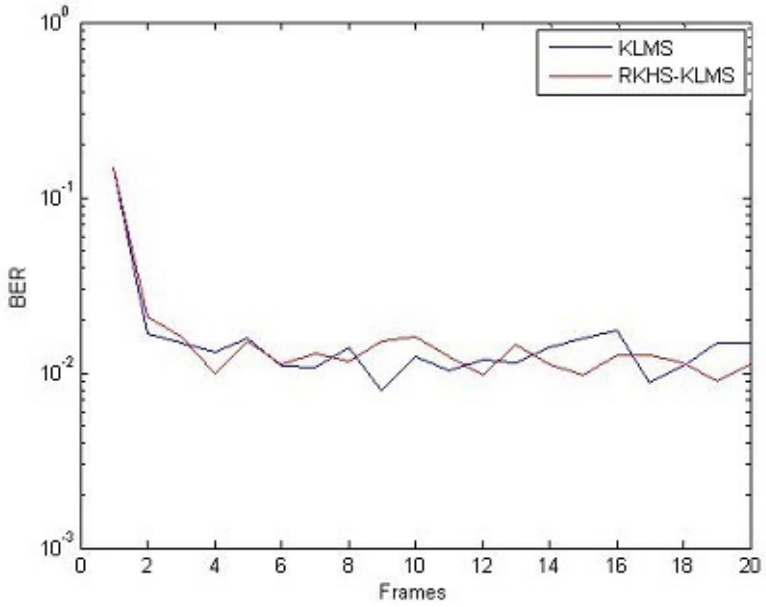

Figura 5.4: Comparación de las probabilidades de error de bit, BER, del KLMS y RKHS-KLMS en una gráfica de 20 tramas, con una $\eta = 0{,}001$, una SNR de 10dB, un tamaño de matriz de kernel de 100 símbolos, una profundidad de memoria de P=5 y un control de dicha profundidad de $\mu = 0{,}9$.

Cómo vemos en la gráfica no observamos cual es mejor o peor a lo largo de las tramas, ya que en algunas tramas es mejor el RKHS-KLMS porque tiene menor probabilidad de error de bit, y en otras tramas es mejor el KLMS no recursivo ya que tiene menor probabilidad error de bit. No podemos afirmar que se obtiene una mejoría.

Los siguientes resultados que representaremos son los obtenidos con el tamaño de la matriz de kernel igual a 150 símbolos y serán mostrados en la Figura 5.5.

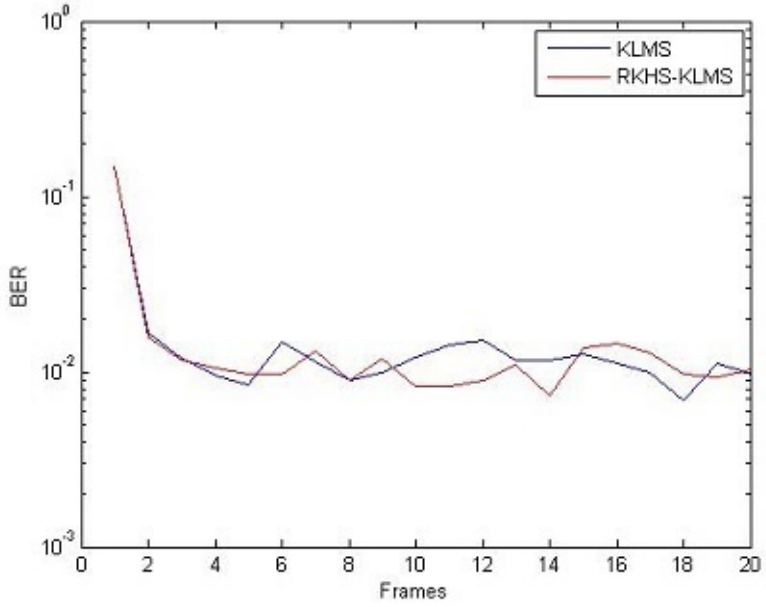

Figura 5.5: Comparación de las probabilidades de error de bit, BER, del KLMS y RKHS-KLMS en una gráfica de 20 tramas, con una $\eta = 0{,}001$, una SNR de 10dB, un tamaño de matriz de kernel de 150 símbolos, una profundidad de memoria de P=5 y un control de dicha profundidad de $\mu = 0{,}9$.

En la gráfica anterior si vemos una leve mejoría que en la figura 5.4, ya que en la mayoría de las tramas la probabilidad de error de bit del KLMS recursivo es inferior que la que obtenemos con el KLMS no recursivo. Por lo que empezamos a ver una leve mejoría.

A continuación, representaremos los resultados obtenidos con el tamaño de la matriz de kernel igual a 200 símbolos y serán mostrados en la Figura 5.6.

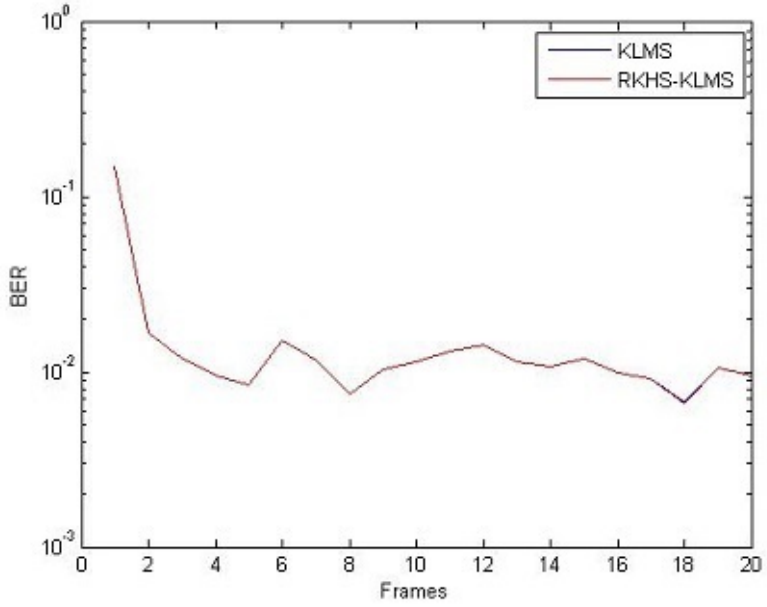

Figura 5.6: Comparación de las probabilidades de error de bit, BER, del KLMS y RKHS-KLMS en una gráfica de 20 tramas, con una $\eta = 0{,}001$, una SNR de 10dB, un tamaño de matriz de kernel de 200 símbolos, una profundidad de memoria de P=5 y un control de dicha profundidad de $\mu = 0{,}9$.

En esta gráfica no se ve que haya mejoría por parte del KLMS recursivo con respecto al no recursivo ya que cómo vemos casi tenemos la misma probabilidad de error de bit en ambos casos. Por eso no hay mejoría alguna.

Ahora representaremos los resultados obtenidos con el tamaño de la matriz de kernel de 250 símbolos y serán mostrados en la Figura 5.7.

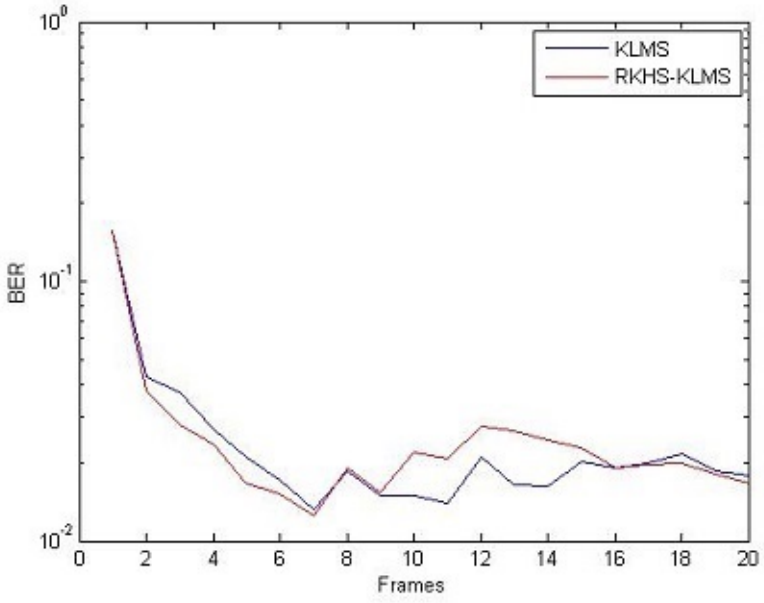

Figura 5.7: Comparación de las probabilidades de error de bit, BER, del KLMS y RKHS-KLMS en una gráfica de 20 tramas, con una $\eta = 0{,}001$, una SNR de 10dB, un tamaño de matriz de kernel de 250 símbolos, una profundidad de memoria de P=5 y un control de dicha profundidad de $\mu = 0{,}9$.

En la gráfica anterior vemos que el algoritmo KLMS recursivo es mejor que el no recursivo, ya que tiene una probabilidad de error menor para las primeras tramas hasta la trama número 7 antes de que se empiecen a descartar datos antiguos. Pero a partir de la trama 7 empieza a tener menor probabilidad de error bit el KLMS no recursivo.

En la siguiente gráfica representaremos los resultados obtenidos con el tamaño de la matriz de kernel de 300 símbolos y serán mostrados en la Figura 5.8.

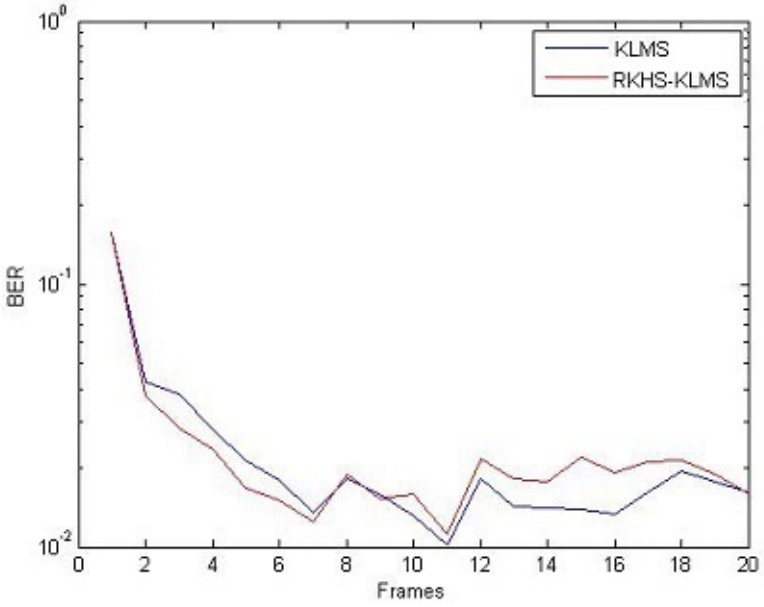

Figura 5.8: Comparación de las probabilidades de error de bit, BER, del KLMS y RKHS-KLMS en una gráfica de 20 tramas, con una $\eta = 0{,}001$, una SNR de 10dB, un tamaño de matriz de kernel de 300 símbolos, una profundidad de memoria de P=5 y un control de dicha profundidad de $\mu = 0{,}9$.

En esta gráfica e igual que en la Figura 5.7, vemos que el algoritmo KLMS con la matriz de kernel recursiva tiene un mejor comienzo ya que tiene menor probabilidad de error que el KLMS no recursivo hasta la trama 7, hasta que el mecanismo de empieza a eliminar los datos más antiguos. Pero a partir de esa trama el KLMS no recursivo tiene mejor probabilidad de error de bit.

Cómo vemos en las gráficas anteriores el KLMS recursivo es mejor para tamaños de la matriz de kernel inferiores a 200 símbolos ya que se obtiene una menor probabilidad de error de bit media que en el caso del KLMS no recursivo. Pero si usamos tamaños de esta matriz mayores a 200 símbolos vemos que tiene una probabilidad de error de bit menor el caso no recursivo que el recursivo.

Además observamos que la probabilidad de error de bit aumenta en cuanto utilizamos un tamaño mayor que 200 símbolos para ambos casos, el KLMS no recursivo y el KLMS recursivo.

5.7. Experimento 4

Representaremos los resultados del experimento, comparando el caso no recursivo (KLMS) con el caso recursivo (RKHS-KLMS) mediante la probabilidad de error de bit variando el parámetro de la profundidad de memoria de la matriz de kernel. Representaremos las probabilidades de error de bit con el transcurso de las tramas para cada valor de P, que va desde 2 hasta 5.

Empezaremos representando los resultados de las probabilidades de error de bit del algoritmo KLMS no recursivo y del KLMS recursivo en función de las tramas transcurridas cuando utilizamos el parámetro $P = 2$ en la Figura 5.9.

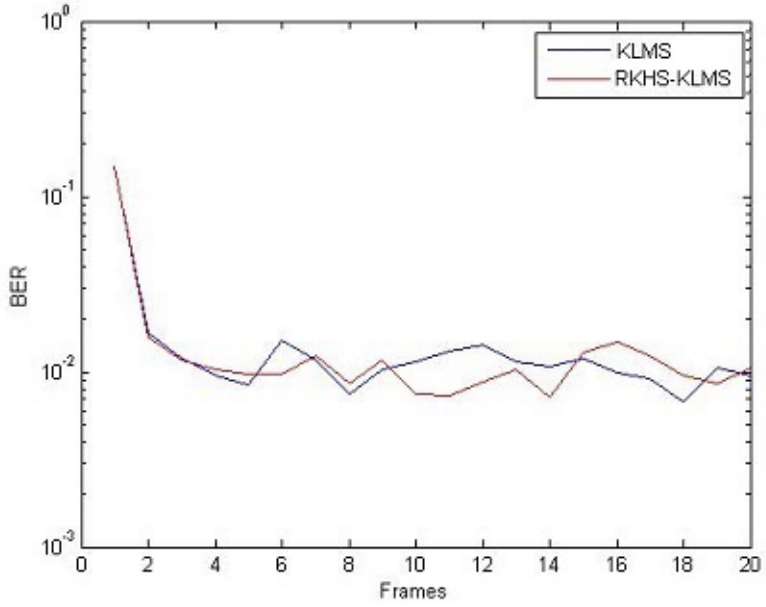

Figura 5.9: Comparación de las probabilidades de error de bit, BER, del KLMS y RKHS-KLMS en una gráfica de 20 tramas, con una $\eta = 0{,}001$, una SNR de 10dB, un tamaño de matriz de kernel de 200 símbolos, una profundidad de memoria de P=2 y un control de dicha profundidad de $\mu = 0{,}9$.

En la gráfica no vemos que el algoritmo KLMS recursivo en este experimento tenga menor probabilidad de error de bit para todas las tramas, si para algunos intervalos de tramas pero para otros intervalos el KLMS no recursivo obtiene menor probabilidad de error de bit. Por lo tanto, no vemos que sea mejor el caso recursivo que el caso no recursivo.

En la segunda gráfica representaremos los resultados de las probabilidades de error de bit del algoritmo KLMS no recursivo y del KLMS recursivo en función de las tramas transcurridas, cuando utilizamos el parámetro $P = 3$ y lo mostraremos en la Figura 5.10.

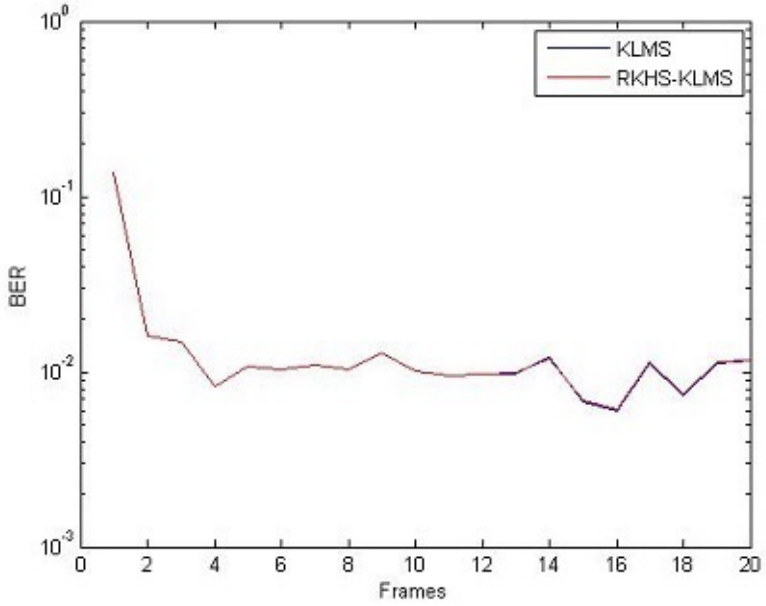

Figura 5.10: Comparación de las probabilidades de error de bit, BER, del KLMS y RKHS-KLMS en una gráfica de 20 tramas, con una $\eta = 0{,}001$, una SNR de 10dB, un tamaño de matriz de kernel de 200 símbolos, una profundidad de memoria de P=3 y un control de dicha profundidad de $\mu = 0{,}9$.

En esta gráfica vemos que no hay diferencias de probabilidad de error de bit entre el caso recursivo y el caso no recursivo, ya que se obtiene aproximadamente la misma probabilidad de error de bit en ambos casos para todas las tramas generadas.

En la siguiente gráfica representaremos los resultados de las probabilidades de error de bit del algoritmo KLMS no recursivo y del KLMS recursivo en función de las tramas transcurridas, cuando utilizamos el parámetro de profundidad de memoria de $P = 4$ y lo mostraremos en la Figura 5.11.

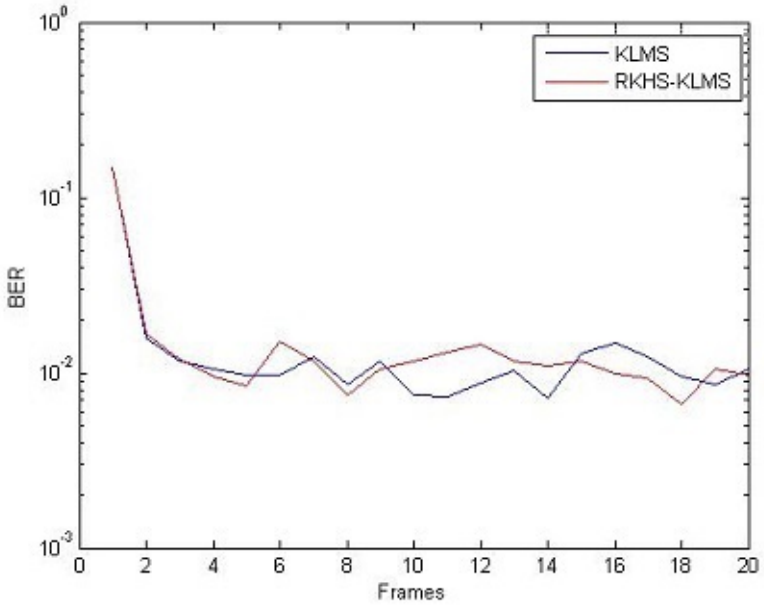

Figura 5.11: Comparación de las probabilidades de error de bit, BER, del KLMS y RKHS-KLMS en una gráfica de 20 tramas, con una $\eta = 0{,}001$, una SNR de 10dB, un tamaño de matriz de kernel de 200 símbolos, una profundidad de memoria de P=4 y un control de dicha profundidad de $\mu = 0{,}9$.

En esta tercera gráfica ocurre lo mismo que en la primera gráfica de este experimento en el que vimos que había intervalos de tramas en los que el algoritmo KLMS recursivo obtenía menor probabilidad de error de bit que el caso no recursivo, pero también había intervalos de tramas en los que el que obtenía menor probabilidad de error de bit es el KLMS no recursivo. Por lo que no podemos decir que se obtenga mejoría del caso recursivo con respecto al caso no recursivo para $P = 4$.

En la última gráfica de este experimento, representaremos los resultados de las probabilidades de error de bit del algoritmo KLMS no recursivo y del KLMS recursivo en función de las tramas transcurridas cuando utilizamos el parámetro $P = 5$ de la matriz de kernel y lo mostraremos en la Figura 5.12.

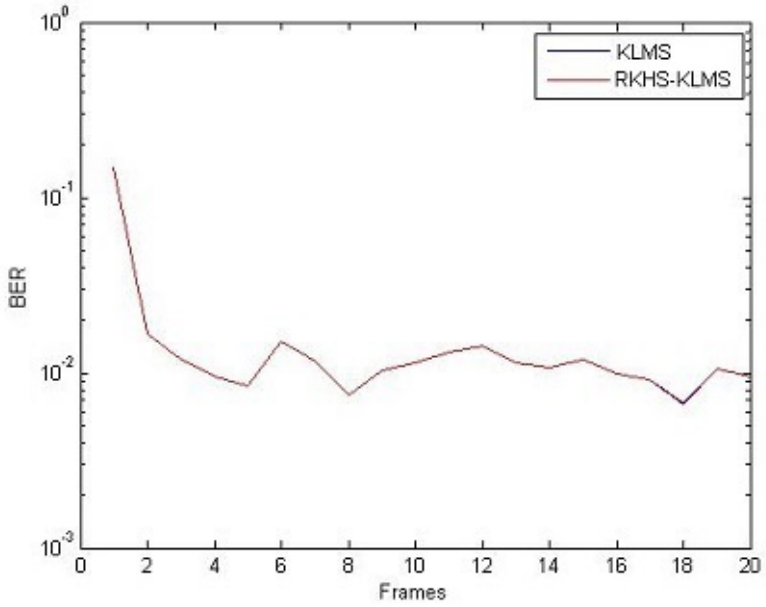

Figura 5.12: Comparación de las probabilidades de error de bit, BER, del KLMS y RKHS-KLMS en una gráfica de 20 tramas, con una $\eta = 0{,}001$, una SNR de 10dB, un tamaño de matriz de kernel de 200 símbolos, una profundidad de memoria de P=5 y un control de dicha profundidad de $\mu = 0{,}9$.

En la gráfica vemos que no hay apenas diferencia entre la curva que representa la probabilidad de error de bit del algoritmo RKHS-KLMS y la que representa dicha probabilidad del algoritmo KLMS no recursivo. Por lo que podemos decir que el caso recursivo no mejora y no obtiene menor probabilidad de error de bit que el caso no recursivo.

Cómo observamos en las gráficas la probabilidad de error de bit tanto para el caso recursivo cómo el no recursivo es igual para ambos casos y además es constante cuando variamos el parámetro P.

5.8. Experimento 5

Representaremos los resultados del experimento 5, comparando el caso no recursivo (KLMS) con el caso recursivo (RKHS-KLMS) mediante sus probabilidades de error de bit promediadas cuando variamos los valores del parámetro SNR al generar los datos que son simulados, lo representamos en la Figura 5.13.

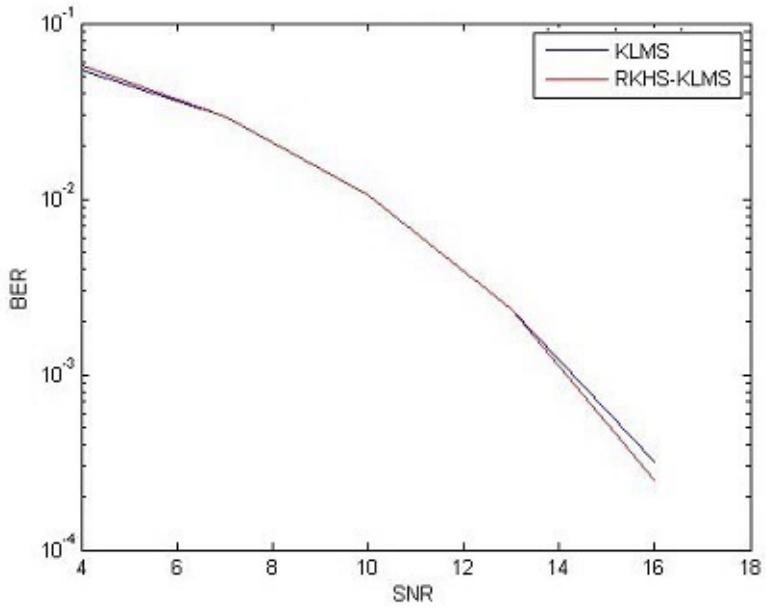

Figura 5.13: Comparación de las probabilidades de error de bit, BER, del KLMS y RKHS-KLMS cuando variamos el valor de la SNR, con una $\eta = 0{,}001$, un tamaño de matriz de kernel de 200 símbolos, una profundidad de memoria de P=5 y un control de dicha profundidad de $\mu = 0{,}9$.

En esta gráfica, vemos cómo varia la probabilidad de error de bit para cada valor de SNR y vemos que para SNR muy pequeñas el KLMS no recursivo tiene la probabilidad de error de bit menor que el caso recursivo. Pero para SNR altas es el KLMS recursivo es el que tiene menor probabilidad de error de bit que el caso no recursivo. Por lo que el KLMS recursivo es mejor para SNRs más altas.

También podemos ver que al aumentar el valor de la SNR, la probabilidad de error de bit promediada para ambos casos disminuye de forma exponencial y para los últimos valores de la SNR no se pueden calcular las probabilidades de error de bit porque son muy bajas.

5.9. Experimento 6

Representaremos los resultados del experimento 6, comparando el caso no recursivo (KLMS) con el caso recursivo (RKHS-KLMS) mediante sus probabilidades de error bit con el paso de las tramas en un entorno no estacionario, que representamos en la Figura 5.14.

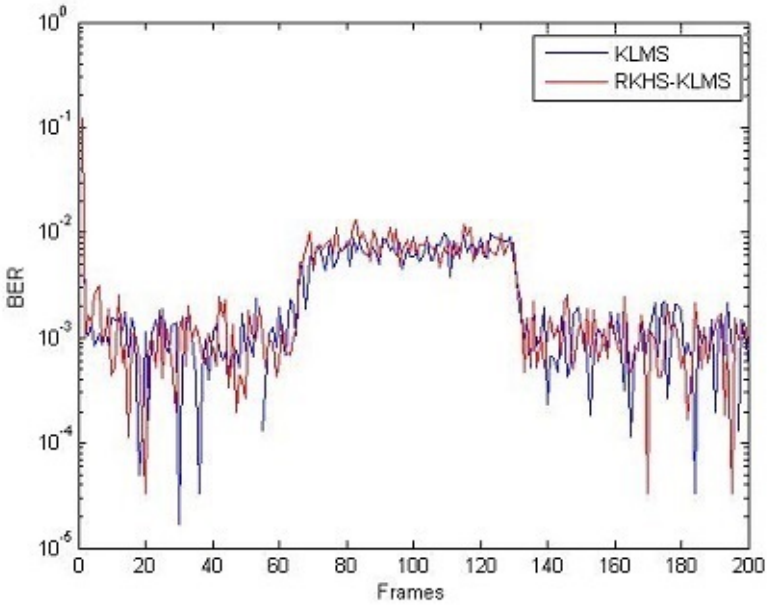

Figura 5.14: Comparación de las probabilidades de error de bit en un entorno no estacionario, BER, del KLMS y RKHS-KLMS en una gráfica de 200 tramas, con una $\eta = 0{,}001$, una SNR de 10dB, un tamaño de matriz de kernel de 200 símbolos, una profundidad de memoria de P=5 y un control de dicha profundidad de $\mu = 0{,}9$.

En esta gráfica, vemos que el utilizar la matriz de kernel recursivo con el algoritmo KLMS no obtenemos ninguna mejoría con respecto a utilizar el KLMS no recursivo, ya que cómo vemos en la gráfica obtenemos aproximadamente la misma probabilidad de error de bit y la misma inestabilidad, debido a que la probabilidad de error varía mucho en ambos casos.

Además, vemos que cuando las señales transmitidas por los usuarios pierden potencia entre las tramas 86 y 130, el KLMS recursivo y el KLMS no recursivo se adaptan de la misma forma aumentando bruscamente la probabilidad de error bit, pasando de una probabilidad de error aproximada de $1 \cdot 10^{-3}$ a una probabilidad aproximada de $1 \cdot 10^{-2}$. Pero que luego recuperan de igual forma en ambos casos y al mismo tiempo que los usuarios recuperan la potencia de sus señales.

Por lo tanto, no vemos que se obtenga mejoría del caso recursivo con respecto al no recursivo al ser usados en un entorno no estacionario.

5.10. Experimento 7

Este ejemplo reproduce el resultado publicado en articulo [16], e ilustra la comparación de distintas estrategias en las que se utiliza como base el algoritmo RLS en su versión kernel. El escenario utilizado es el no estacionario ya descrito anteriormente. En particular, se compara el algoritmo KRLS que utiliza como base el kernel recursivo, con los algoritmos KLMS presentado en libro [13], KRLS del libro [7], junto con sus versiones dispersas mediante el método 'surprise criterion' (SC) del libro [12] usando el kernel estándar RBF. El factor de regularización es $\eta = 0{,}001$, el factor de olvido 0.99. El kernel recursivo usó $\mu = 0{,}9$. El experimento se repitió 10 veces. Y se representará en la Figura 5.15.

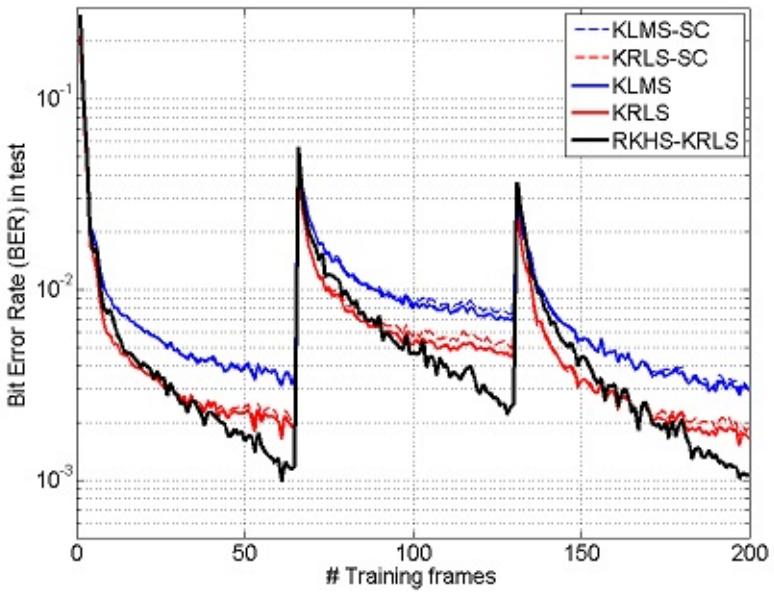

Figura 5.15: Bit Error Rate (BER) con respecto al tiempo para el usuario deseado.

EL kernel recursivo tuvo el mismo tiempo computacional que el resto de kernels. El tiempo de optimización es $O(t^2)$, siendo el mismo que el del KRLS. Como puede verse, el algoritmo que utiliza el kernel recursivo tiene ventaja sobre el resto. Puede verse también que el algoritmo KLMS es capaz de seguir las variaciones del canal, aunque este resultado no ha podido reproducirse en los experimentos anteriores utilizando algoritmos KLMS.

CAPÍTULO 6

CONCLUSIONES

En este proyecto fin de carrera se presenta una metodología para desarrollar un kernel de Mercer recursivo en un RKHS, donde la aplicación escogida consiste en procesado de arrays no lineal. El método de recursión está basado en el filtro gamma y fue publicado en [16].

El kernel, tal como se desarrolla, puede ser incluido en cualquier algoritmo que admita el uso de kernels, con la particularidad de que la recursión produce no una matriz de kernels, sino un número arbitrario de ellas, que puede ser escogido por el usuario. La recursión necesita que se defina como kernel básico cualquier Kernel que cumpla con el Teorema de Mercer, montándose la recursión a partir de este. En el caso particular de que el número de matrices sea una, la recursión se simplifica a ese kernel básico. Se ha probado en [16] que este kernel cumple con el teorema de Mercer si lo hace el kernel básico utilizado.

La particularidad que se pretende explotar en esta aplicación es el hecho de que la información contenida en la primera matriz se expande con el tiempo a las otras matrices en posiciones contiguas a cada posición de la primera. Es decir la información que está contenida en una posición de la primera matriz, se expande en el siguiente instante de tiempo a la siguiente matriz y a las posiciones contiguas de esta, y sucesivamente a las siguientes matrices. De esta forma, si para limitar el coste computacional del algoritmo, la matriz se trunca mediante la eliminación de una fila y una columna, no se destruye con ello toda la información de la muestra que genera esa fila y esa columna. Si esa información es relevante para la tarea, con ello se pretende minimizar el deterioro en las prestaciones del algoritmo.

En este proyecto se han probado los siguientes algoritmos: el KLMS, el KRLS, el RKHS-KLMS y el RKHS-KRLS.

El algoritmo KLMS utiliza el algoritmo lineal LMS en un espacio transformado no lineal, el espacio de Hilbert. Este algoritmo es el más sencillo de los dos algoritmos y lo que busca es minimizar el error cuadrático medio (MSE) del sistema, y para ello lo primero que hace es calcular el error en cada instante de tiempo n y luego actualiza el vector de pesos $\mathbf{w}$, calculando $\mathbf{w}[n+1]$, para posteriormente calcular el error en el siguiente instante de tiempo.

El algoritmo KRLS deriva del algoritmo RLS aplicado al espacio transformado de kernel de Hilbert (RKHS) y también tiene como objetivo minimizar el error cuadrático medio del sistema en cada instante de tiempo n al igual que se hace para el algoritmo KLMS pero para el algoritmo KRLS no sólo actualizamos el peso actual, sino que también actualizamos los pesos anteriores que ya teníamos.

Los algoritmos RKHS-KLMS y RKHS-KRLS, son los algoritmos KLMS y KRLS respectivamente pero que en lugar de aplicarse sobre el espacio de kernel de Hilbert directamente, se aplican al filtro de kernel recursivo que actúa sobre el espacio de kernel de Hilbert. Este filtro consiste en un conjunto de matrices limitadas a un tamaño máximo de datos de entrada y el número de matrices está determinado por un parámetro de profundidad P. Los datos se van introduciendo poco a poco en la matriz principal y se generan datos recursivos de los datos anteriores que se almacenan en resto de las matrices P-1. Cuando dicha matriz alcanza su máximo se empiezan a descartar los datos más antiguos y es aquí donde empiezan a ser útiles las matrices recursivas, ya que proporcionan información de datos anteriores incluso los que ya se han eliminado por el algoritmo de descarte utilizado.

Los escenarios que hemos utilizado consistieron en dos tipos de escenario: un escenario estacionario en el que las potencias de las señales no variaban con el transcurso de las tramas y eran iguales a 1. Y un escenario no estacionario en el que la potencia de las señales estaba al máximo y en un intervalo de tramas, la potencia de la señal disminuía un 20 % de la potencia máxima y después las señales recuperaban su potencia máxima.

Se han presentado resultados en diversos experimentos realizados. Se han hecho experimentos en los que hemos visto cómo se comportaba el algoritmo KLMS en el caso recursivo y en el caso no recursivo, al variar varios parámetros como son el tamaño de la matriz de kernel, el parámetro de profundidad de la matriz de kernel P y al variar el relación señal ruido de las señales en un entorno estacionario. Y también hemos visto el comportamiento de los algoritmos KLMS y KRLS, en sus casos recursivos y no recursivos, en un entorno no estacionario.

En el artículo citado se presenta evidencia de la ventaja del kernel recursivo con respecto a otros kernels debido a la mayor robustez de este kernel frente a procesos de poda. Estos resultados se han basado en pruebas realizadas fundamentalmente con algoritmos KRLS. En este proyecto fin de carrera, no obstante, estos resultados no han podido ser reproducidos para algoritmos tipo KLMS. Ello puede deberse a diversas circunstancias que quedan fuera del alcance de este proyecto. La primera posibilidad puede deberse al hecho de que el algoritmo LMS no es consistente en varianza, produciendo un error superior al de los algoritmos RLS. Esto por sí solo puede enmascarar cualquier mejora. Otra causa puede deberse a una deficiente elección de los parámetros libres, que pueden producir resultados subóptimos. No obstante, un estudio exhaustivo de estos parámetros está fuera de los límites de este proyecto, dadas las limitaciones del hardware disponible para él.

Entre los trabajos futuros de este proyecto está un estudio de los parámetros óptimos de este algoritmo, así como un estudio de diferentes kernels base para aplicaciones de comunicaciones.

BIBLIOGRAFÍA

[1] N. Aronszajn, *"Theory of reproducing kernels"*, Transactions of the American Mathematical Society, vol. 68, pp. 337-404, 1950.

[2] C. J. C. Burges, *"A tutorial on support vector machines for pattern recognition"*, Data Mining and Knowledge Discovery, vol. 2, pp. 121-167, 1998.

[3] I. Cha and S. Kassam, *"Channel equalization using adaptive complex radial basis function networks"*, IEEE Journal on Selected Areas in Communications, vol. 13, pp. 122-131, 1995.

[4] D. Erdogmus , D. Rende , J. C. Principe and T. F. Wong, *"Nonlinear channel equalization using multilayer perceptrons with information - theoretic criterion"*, In Proceedings of IEEE Workshop on Neural Networks for Signal Processing, pp. 443-451, 2001.

[5] D. Gabor , W. P. L. Wilby and R. Woodcock, *"A universal non-linear filter, predictor and simulator which optimizes itself by a learning process"*, In Proceedings of the Institution of Electrical Engineers, vol. 108, pp. 422-435, 1960.

[6] D. Tuia, G. Camps-Valls and M. Martínez-Ramón, *"Explicit recursivity into reproducing kernel Hilbert spaces"*, IEEE-ICASSP, pp. 4148-4151, 2011.

[7] Y. Engel, S. Mannor and R. Meir, *"The Kernel Recursive Least Squares Algorithm"*, IEEE Transactions on Signal Processing, vol. 52, pp. 2275-2285, 2004.

[8] M. O. Franz and B. B. Schölkopf, *"A unifying view of Wiener and Volterra thory and polynomial kernel regression"*, Neural Computation, vol. 18, pp. 3097-3118, 2006.

[9] G. Kechriotis, E. Zarvas and E. S. Manolakos, *"Using recurrent neural networks for adaptive communication channel equalization"*, IEEE Transactions on Neural Networks, vol. 5, pp. 267-278, 1994.

[10] I. Goethals, K. Pelckmans, J. A. K. Suykens and B. De Moor, *"Subspace Identification of Hammerstein System Using Least Squares Support Vector Machines"*, IEEE Transactions on Automatic Control, vol. 50, no. 10, pp. 1509-1519, 2005.

[11] J.C. Principe, B. de Vries, and P. Guedes de Oliveira, *"The gamma filter - a new class of adaptative IIR filters with restricted feedback"*, IEEE Transactions on Signal Processing, vol. 41, pp. 649-656, 1993.

[12] J.C. Principe, W. Liu and Il. Park , *"An Information Theoretic Approach of Designing Sparse Kernel Adaptive Filters"*, IEEE Transactions on Neural Networks, vol. 20, no. 12, pp. 1950-1961, 2009.

[13] J.C. Principe, W. Liu and P.P. Pokharel, *"The Kernel Least-Mean-Square Algorithm"*, IEEE Transactions on Signal Processing, vol. 56, no. 2, pp. 543-554, 2008.

[14] M. Martínez-Ramón, J.L. Rojo-Álvarez, G. Camps-Vals, A. Navia Vázquez, E. Soria-Olivas and A.R. Figueiras-Vidal, *"Support vector machines for nonlinear kernel ARMA sysyem identification"*, IEEE Transactions on Neural Networks, vol. 17, pp. 1617-1622, 2006.

[15] S. Haykin, A. H. Sayed, J. R. Zeidler, P. Yee, and P. C. Wei, *"Adaptive tracking of linear time - variant systems by extended RLS algorithms"*, IEEE Transactions on Signal Processing, vol. 45, pp. 1118-1128, 1997.

[16] D. Tuia and J. Munoz-Mari and J.L. Rojo-Alvarez and M. Martínez-Ramón and G. Camps-Valls, *"Explicit Recursive and Adaptive Filtering in Reproducing Kernel Hilbert Spaces"*, IEEE Transactions on Neural Networks and Learning Systems, En prensa, 2014.

[17] W. Liu, J. C. PRINCIPE and S. Haykin, *"Kernel Adaptive Filtering"*, Wiley, 2010.

[18] B. Widrow and M.E. Hoff, *"Adaptive switching circuits"*, IRE WESCON Convention Record, vol. 4, pp. 96-104, 1960.

[19] A. Navia-Vázquez, M. Martínez-Ramón, L.E. García-Muñoz and C.G. Christodoulou, *"Aproximate Kernel Orthogonalization for Antenna Array*

Processing", IEEE Transactions on Antennas and Propagation, vol. 58, no. 12, pp. 3942-3950, 2010.